Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

К. В. Двірничук, Д. О. Вацек

ВЕБ-ПРОГРАМУВАННЯ ТА ВЕБ-ДИЗАЙН

Навчальний посібник

Чернівці
Чернівецький національний університет
імені Юрія Федьковича
2022

УДК 004.774.6 (075.8)
Д 239

Друкується за ухвалою редакційно–видавничої ради
Чернівецького національного університету
імені Юрія Федьковича

Рецензенти:
Ісарюк І. В., кандидат фізико-математичних наук,
викладач комп'ютерних дисциплін
(Чернівецький кооперативний
фаховий коледж економіки і права);
Артеменко О. І., кандидат технічних наук, доцент
(ПВНЗ Буковинський університет).

Двірничук К. В., Вацек Д. О.
Д 239 Веб-програмування та веб-дизайн **:** навч. посіб. Чернівці : Чернівец. нац. ун-т ім. Ю. Федьковича, 2022. 472 с.

ISBN

У навчальному посібнику наведено основні теоретичні відомості мови гіпертекстової розмітки HTML, таблиць каскадних стилів CSS, фреймворку Bootstrap, мов програмування JavaScript та PHP, js-бібліотеки jQuery, технології AJAX, необхідні для виконання лабораторних завдань із дисциплін «WEB-програмування та дизайн», «Технологія розробки Front-end» для студентів спеціальності «Комп'ютерна інженерія» ОПП «Комп'ютерна інженерія» та «Програмування мобільних і вбудованих комп'ютерних систем та засобів Інтернету речей».

УДК 004.774.6 (075.8)

ISBN

ЗМІСТ

ВСТУП

На сьогоднішній день велику частину всієї програмної розробки складає саме **веб-розробка**, і з кожним днем її значимість зростає. Така тенденція спостерігається завдяки глобальній інформатизації та наданні всеможливих послуг і товарів в мережі Інтернет. Це дає змогу географічно розширити можливості економіки, спростити ведення бізнесу, надавати користувачам нові можливості в Україні та світі. Реалізація цих можливостей відбувається через вузли інтернет-мережі – **веб-сайти**.

Веб-сайти відображаються за допомогою веб-браузерів. Це означає, що відображення вмістимого сайтів залежить не від операційних систем, а тільки від браузерів та їхніх версій. Тому всі веб-додатки можна назвати кросплатформними програмними продуктами.

Веб-сайти, в свою чергу, складаються з «frontend»- та «backend»-частин, тобто клієнтської та серверної частин. Сама розробка сайтів, які можуть бути як статичними, так і динамічно взаємодіяти з користувачем, складається **з веб-дизайну та веб-програмування**. Розглянемо більш детально ці поняття.

Веб-дизайн – галузь веб-розробки та різновид дизайну, завданням якого є проектування користувацьких інтерфейсів для веб-додатків. Веб-дизайнери проектують логічну структуру сторінок веб-сайту; продумують найзручніші способи подання інформації; займаються художнім оформленням веб-проекту. Технологіями створення веб-дизайну є **HTML**, **CSS** та інструменти створення адаптивної верстки **GridLayout, Flexbox** (для коректного відображення веб-сторінок на будь-яких гаджетах). Останні детально описані в посібнику.

За допомогою названих технологій, тобто без мов програмування, можна створити повноцінний **статичний сайт** для представлення інформації. Такий сайт являє собою набір статичних html (htm, dhtml, xhtml) сторінок, найчастіше пов’язаних між собою переходами (посиланнями) та логічно пов’язаним наповненням. Вони можуть містити текст, зображення, мати мультимедіа-вміст (аудіо, відео) та HTML-теги. Всі необхідні зміни вносяться у html-код документів

(сторінок) сайту, для чого потрібен доступ до файлів на веб-сервері.

Одним із розділів посібника є короткий огляд **Bootstrap 4**, який являє собою набір інструментів із відкритим кодом, і призначений для розробки веб-додатків. Bootstrap 4 містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Такий набір інструментів спрощує розробку динамічних та статичних веб-сайтів та є, взагалі кажучи, CSS- та JS-фреймворком для веб-дизайну.

Веб-програмування – розділ програмування, орієнтований на розробку веб-додатків (програм, що забезпечують функціонування динамічних сайтів). **Мови веб-програмування** призначені для роботи з веб-технологіями. Мови веб-програмування можна умовно розділити на дві групи, які часто перетинаються: клієнтські та серверні.

Клієнтські мови обробляються на стороні клієнта (frontend), як правило, це виконує браузер (веб-переглядач). Це створює головну проблему клієнтських мов – результат виконання програми (скрипта) залежить від браузера користувача. Тобто, якщо користувач заборонив виконувати клієнтські програми, то вони виконуватися не будуть, хоч би як цього бажав програміст. Крім того, можлива ситуація, коли в різних браузерах або версіях одного і того ж браузера, один і той же скрипт буде виконуватися по-різному. З іншого боку, за рахунок програм, виконуваних на клієнті, програміст може спростити роботу серверних програм, у тих випадках, де це можливо, і зменшити навантаження на сервер.

До клієнтських мов програмування відносять JavaScript, Java і VBScript. Особливої уваги серед вказаних мов заслуговує **мова JavaScript** та технології, побудовані на її основі (React, jQuery, AngularJS, …). Це найпопулярніша клієнтська мова програмування, яку підтримують усі сучасні веб-браузери. За допомогою JavaScript сторінки сайту «оживають», тобто набувають нових інтерактивних, динамічних рис. Саме мові JavaScript та її **бібліотеці jQuery** присвячена значна частина посібника.

Коли користувач виконує запит на будь-яку сторінку (переходить на неї за посиланням або вводить адресу в адресному рядку браузера), ця сторінка спочатку обробляється **на серверній частині** (backend), тобто виконуються всі програми, пов'язані з нею, і тільки потім результат обробки повертається до відвідувача по мережі у вигляді файлу. Цей файл може мати розширення HTML, PHP, ASP, ASPX, Perl, SSI, XML, DHTML, XHTML. Робота таких програм повністю залежить від сервера, на якому розташований сайт, і від того, яка версія мови підтримується. Важливою стороною роботи **серверних мов** є можливість організації безпосередньої взаємодії із системою управління базами даних (**СУБД**) – сервером бази даних, де зберігається вся потрібна інформація.

Серед відомих мов програмування, на стороні сервера, можна виділити Java, PHP, C#, JavaScript (Node.js), Perl, Python та мову запитів SQL до баз даних. Проте найпопулярнішою серверною мовою програмування вважається **PHP**. На її основі розроблені десятки CMF- та CMS-систем.

З погляду веб-програмування, **framework-система (CMF-система)** – це платформа, що дозволяє виконувати завдання, які виникають при створенні інтернет-додатків. **Найпоширенішими PHP-фреймворками є**: Zend Framework, Yii, Phalcon, Codeigniter, Laravel, Symfony.

Розглядаючи поняття framework-системи, не можна обійти стороною поняття **системи управління контентом**. Дуже часто поняття CMF (Content Management Framework) плутають із поняттям **CMS** (Content Management System). Але це принципово різні речі.

CMS-система – це набір модулів для швидкого створення сайтів. На відміну від CMF, CMS-система – це завершений продукт, який орієнтований насамперед не на програмістів, а на користувачів, не знайомих із тонкощами створення інтернет-додатків. CMS-система («движок сайту») дозволяє за лічені години створити сайт, який складається з обмеженого набору готових модулів. CMS-системи розробляються переважно без врахування їх подальшого росту. В підсумку, відсутність жорсткої внутрішньої архітектури в CMS-системах істотно ускладнює процес впроводження проекту. **Найпоширенішими**

CMS-системами є Joomla, Drupal, WordPress, OctoberCMS, Opencart, Magento.

Саме мові програмування PHP, найпоширенішій мові програмування з найбільшою кількістю CMF- та CMS-систем, присвячена значна частина посібника.

Пропонований посібник може бути використаний при вивченні курсів «WEB-програмування та дизайн», «Технологія розробки Front-end» спеціальності «Комп'ютерна інженерія» галузі «Інформаційні технології» за освітніми програмами «Комп'ютерна інженерія» та «Програмування мобільних і вбудованих комп'ютерних систем та засобів Інтернету речей».

Курс «WEB-програмування та дизайн» включає вивчення технологій створення веб-дизайну, тобто технологій розробки Front-end. Тому даний посібник може використовуватися при вивченні курсу «Технологія розробки Front-end».

1. РОБОЧА ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ «WEB-ПРОГРАМУВАННЯ ТА ДИЗАЙН»

1.1. Анотація дисципліни

Курс «WEB-програмування та дизайн» призначений для формування навичок створення веб-сайтів, а саме створення html-сторінок, css-стилів, адаптивного дизайну, js- та php-скриптів, створення та редагування таблиць баз даних MySQL засобами мови PHP, відправка даних форми без перевантаження веб-сторінки (AJAX), створення анімацій із використанням бібліотеки jQuery.

Курс «WEB-програмування та дизайн» є узагальнюючим та розширюючим для курсу «Технологія розробки Front-end», оскільки містить не тільки способи розробки Front-end, а й технології розробки Back-end. Тому посібник побудовано на основі робочої навчальної програми дисципліни «WEB-програмування та дизайн», як такої, що включає в себе і робочу навчальну програму курсу «Технологія розробки Front-end».

Мета навчальної дисципліни: надати студентам систематизовані знання мов програмування PHP, JavaScript, js-бібліотеки jQuery, фреймворку Bootstrap, мови гіперрозмітки HTML, таблиці каскадних стилів CSS. Зокрема, знання про синтаксис мов і технологій, об'єктно-орієнтовані принципи, вміння працювати з рядками, текстовими файлами, базами даних, вміння створювати html-форми, обробляти їх та отримувати дані з них php-скриптами, після чого записувати дані в базу даних.

Завдання – на основі отриманих теоретичних знань та практичних навичок виробити у студентів вміння створювати власні повноцінні багатофункціональні сайти з адаптивним дизайном, які можуть бути представлені веб-сторінками з html-формами, текстовою інформацією, графічними даними та бути зв'язані з базами даних MySQL засобами PHP.

Пререквізити. Для коректного розуміння і засвоєння матеріалу цього курсу слухачі повинні попередньо пройти курси: «Програмування Ч1. Основи алгоритмізації та

програмування», «Програмування Ч2. Програмування мовою С++», «Програмування Ч3. Основи об'єктно-орієнтованого програмування».

1.2. Результати навчання

У результаті вивчення навчальної дисципліни студент повинен

знати: завдання та принципи мов програмування PHP, JavaScript, js-бібліотеки jQuery, фреймворку Bootstrap, мови гіперрозмітки HTML, таблиці каскадних стилів CSS, зокрема, синтаксис мов та технологій, об'єктно-орієнтовані принципи, вміння працювати з рядками, текстовими файлами, базами даних, вміння створювати html-форми, обробляти їх та отримувати дані з них php-скриптами, після чого записувати дані в БД;

вміти: створювати власні повноцінні багатофункціональні сайти з адаптивним дизайном, які можуть бути представлені веб-сторінками з html-формами, текстовою інформацією, графічними даними та бути зв'язані з базами даних MySQL засобами PHP.

Набути компетентностей:

З – загальних

З3. Здатність застосовувати знання у практичних ситуаціях.

З7. Вміння виявляти, ставити та вирішувати проблеми.

З8. Здатність працювати в команді.

Р – фахових

Р2. Здатність використовувати сучасні методи і мови програмування для розробки алгоритмічного та програмного забезпечення.

Р3. Здатність створювати системне та прикладне програмне забезпечення комп'ютерних систем і мереж.

Р11. Здатність оформляти отримані робочі результати у вигляді презентацій, науково-технічних звітів.

Р15. Здатність аргументувати вибір методів розв'язування спеціалізованих задач, критично оцінювати отримані результати, обґрунтовувати та захищати прийняті рішення.

ПРН - програмовані результати навчання за загальними та загально-професійними фаховими компетентностями

N1. Знати та розуміти наукові положення, що лежать в основі функціонування комп'ютерних засобів, систем і мереж.

N2. Мати навички проведення експериментів, збирання даних та моделювання в комп'ютерних системах.

N6. Вміти застосовувати знання для ідентифікації, формулювання і розв'язування технічних задач спеціальності, використовуючи методи, що є найбільш придатними для досягнення поставлених цілей.

N7. Вміти розв'язувати задачі аналізу та синтезу засобів, характерних для спеціальності.

N8. Вміти системно мислити та застосовувати творчі здібності до формування нових ідей.

N9. Вміти застосовувати знання технічних характеристик, конструктивних особливостей, призначення і правил експлуатації програмно-технічних засобів комп'ютерних систем та мереж для вирішення технічних задач спеціальності.

N10. Вміти розробляти програмне забезпечення для вбудованих і розподілених застосувань, мобільних і гібридних систем, розраховувати, експлуатувати типове для спеціальності обладнання.

N13. Вміти ідентифікувати, класифікувати та описувати роботу комп'ютерних систем та їх компонентів.

N14. Вміти поєднувати теорію і практику, а також приймати рішення та виробляти стратегію діяльності для вирішення завдань спеціальності з урахуванням загальнолюдських цінностей, суспільних, державних та виробничих інтересів.

N15. Вміти виконувати експериментальні дослідження за професійною тематикою.

N16. Вміти оцінювати отримані результати та аргументовано захищати прийняті рішення.

N20. Усвідомлювати необхідність навчання впродовж усього життя з метою поглиблення набутих та здобуття нових фахових знань, удосконалення креативного мислення.

1.3. Опис навчальної дисципліни. Загальна інформація

Загальна інформація

Назва навчальної дисципліни WEB-програмування і дизайн												
Форма	Рік підготовки	Семестр	Кількість			Кількість годин						Вид підсумкового контролю
			кредитів	годин	зміст. мод.	лекції	практичні	семінарські	лабораторні	сам. роб.	інд. завд.	
Денна	2	3	4	120	2	30	-	-	30	60	-	іспит
Заочна	2	3	4	120	2	8	-	-	8	104	-	іспит

Примітка. Співвідношення кількості годин аудиторних занять до самостійної і індивідуальної роботи становить:
для денної форми навчання – 1,0 ((30+30)/60);
для заочної форми навчання – 0,15 ((8+8)/104).

Дидактична карта навчальної дисципліни

Назви змістових модулів і тем	Кількість годин											
	Денна форма						Заочна форма					
	∑	у тому числі					∑	у тому числі				
		л	п	лаб	інд	с.р.		л	п	лаб	інд	с.р.
Теми	**Змістовий модуль 1. WEB-дизайн**											
Тема 1. Мова гіпер-розмітки HTML	4	2		2			4	1		1		2
Тема 2. Таблиці каскадних стилів CSS	12	6		6			12	1		1		10
Тема 3. Фреймворк Bootstrap	4	2		2			4	1		1		2
Разом за модулем 1	**20**	**10**		**10**			**20**	**3**		**3**		**14**

Назви змістових модулів і тем	Кількість годин											
	Денна форма						Заочна форма					
	∑	у тому числі					∑	у тому числі				
		л	п	лаб	інд	с.р.		л	п	лаб	інд	с.р.
Теми	**Змістовий модуль 2. WEB-програмування**											
Тема 4. Мова програму-вання JavaScript	36	8		8		30	39	2		2		35
Тема 5. Бібліотека jQuery	4	2		2			12	1		1		10
Тема 6. Мова програмування PHP	36	8		8		30	39	2		2		35
Тема 7. Технологія AJAX	4	2		2			10					10
Разом за модулем 2	**80**	**20**		**20**		**60**	**100**	**5**		**5**		**90**
Усього годин	**120**	**30**		**30**		**60**	**120**	**8**		**8**		**104**

Теми семінарських або практичних, або лабораторних занять

№ п/п	Назва теми	Кількість годин
1	Мова гіперрозмітки HTML	1
2	Таблиці каскадних стилів (CSS)	2
3	Модулі CSS3: FlexBox та Grid Layout	2
4	Фреймворк Bootstrap	2
5	Валідація форм засобами JavaScript	2
6	Онлайн-калькулятор на JavaScript	3
7	Пошук на веб-сторінці засобами JS	2
8	Плагін jQuery	3
9	Авторизація доступу до сайту (PHP)	2

10	Запис даних з HTML-форми у БД MySQL	3
11	Адмін-панель на PHP	5
12	Технологія AJAX	3
	РАЗОМ	**30**

Тематика індивідуальних завдань. У даному курсі виконання індивідуальних завдань не передбачено. ІНДЗ може бути рекомендовано в окремих випадках для студентів, які успішно засвоїли основний навчальний матеріал, із метою поглибленого вивчення чи удосконалення навичок певного змістового модуля, або в цілому для навчальної дисципліни за рішенням кафедри чи викладача.

Самостійна робота

№ п/п	Назва теми	Кількість годин
1	Фреймворк AngularJS	10
2	Платформа Node.js	10
3	JS-бібліотека React	10
4	PHP-фреймворк Yii	15
5	PHP-фреймворк Zend Framework	15

Форми і методи навчання. Форми навчання – це оглядові та проблемні лекції, лабораторні заняття, заняття із застосуванням комп'ютерної та телекомунікаційної техніки, інтерактивні заняття з навчанням одних студентів іншими, інтегровані заняття, проблемні заняття, відеолекції, відеозаняття і відеоконференції засобами Google Meet, Zoom, Cisco Webex, заняття з використанням системи електронного навчання Moodle.

Методи: проблемний виклад матеріалу, частково-пошукові та дослідницькі лабораторні практикуми, презентації, кейс-стаді, консультації і дискусії, робота в інтернет-класі: електронні лекції, лабораторні роботи, дистанційні консультації та ін., спрямовані на активізацію і стимулювання навчально-пізнавальної діяльності студентів.

Підходи до навчання: використовуються студенто-центрований, проблемно-орієнтований, діяльнісний, комунікативний, професійно-орієнтований, міждисциплінарний підходи.

Реалізація навчального процесу здійснюється під час лекційних, лабораторних занять, самостійної позааудиторної роботи з використанням сучасних інформаційних технологій навчання, консультацій із викладачами.

Для формування вмінь та навичок застосовуються такі методи навчання:

– вербальні/словесні (лекція, пояснення, розповідь, бесіда, інструктаж);

– наочні (спостереження, ілюстрація, демонстрація);

– практичні (проведення експерименту, практики);

– пояснювально-ілюстративний або інформаційно-рецептивний, який передбачає подання готової інформації викладачем та її засвоєння студентами;

– репродуктивний (виконання лабораторних завдань за зразком);

– метод проблемного викладу матеріалу на лекційних заняттях.

Технічне й програмне забезпечення/обладнання

Комп'ютери в комп'ютерних класах №307, №311, №312, №313 8к. ЧНУ – кафедра КСМ.

Програмне забезпечення: операційна система (Windows, Mac, Linux), текстовий редактор, веб-сервер (OpenServer, Apache+PHP+MySQL, …), веб-браузер.

1.4. Система контролю та оцінювання

Розподіл максимально можливої кількості балів, які отримують студенти за виконання всіх видів навчальної діяльності.

Змістовий модуль 1. WEB-дизайн

Т1. Мова гіперрозмітки HTML (виконання лабораторної роботи №1 – 5 балів).

Т2. Таблиці каскадних стилів CSS (виконання лабораторних робіт №2,3 – 10 балів).

Т3. Фреймворк Bootstrap (виконання лабораторної роботи №4 – 5 балів).

М1. Модульна контрольна робота №1 – 5 балів.

Змістовий модуль 2. WEB-програмування

Т4. Мова програмування JavaScript (виконання лабораторних робіт №5,6,7 – 15 балів).

Т5. Бібліотека jQuery (виконання лабораторної роботи №8 – 5 балів).

Т6. Мова програмування PHP (виконання лабораторних робіт №9,10,11 – 15 балів).

Т7. Технологія AJAX (виконання лабораторної роботи №12 – 5 балів).

М2. Модульна контрольна робота №2 – 5 балів.

Шкала оцінювання: національна та ЄКТС

<table>
<tr><th rowspan="2">Сума балів за всі види навчальної діяльності</th><th rowspan="2">Оцінка ECTS</th><th colspan="2">Оцінка за національною шкалою</th></tr>
<tr><th>для екзамену, курсового проекту (роботи), практики</th><th>для заліку</th></tr>
<tr><td>90 – 100</td><td>A</td><td>відмінно</td><td rowspan="5">зараховано</td></tr>
<tr><td>80 – 89</td><td>B</td><td rowspan="2">добре</td></tr>
<tr><td>70 – 79</td><td>C</td></tr>
<tr><td>60 – 69</td><td>D</td><td rowspan="2">задовільно</td></tr>
<tr><td>50 – 59</td><td>E</td></tr>
<tr><td>35 – 49</td><td>FX</td><td>незадовільно з можливістю повторного складання</td><td>не зараховано з можливістю повторного складання</td></tr>
<tr><td>0 – 34</td><td>F</td><td>незадовільно з обов’язковим повторним вивченням дисципліни</td><td>не зараховано з обов’язковим повторним вивченням дисципліни</td></tr>
</table>

Засоби оцінювання. Засобами оцінювання результатів навчання студента є: завдання для виконання лабораторних робіт, а також модульні контрольні роботи.

Форми поточного та підсумкового контролю. Формами поточного контролю рівня знань є усна та письмова відповідь студента при захисті виконаних лабораторних робіт, а також письмова відповідь при написанні модульних контрольних робіт. Формами підсумкового контролю рівня знань є усна та письмова відповідь студента при здачі іспиту.

Політика дисципліни. Визначається системою вимог викладача щодо рівня знань і засвоєння матеріалу студентом при вивченні дисципліни та ґрунтується на засадах академічної доброчесності з урахуванням норм законодавства України щодо академічної доброчесності та Статуту, положень Університету, й інших нормативних документів, які регламентують організацію освітнього процесу при вивченні дисципліни.

Вимоги стосуються заохочень і нарахування додаткових балів за активну участь у дискусіях щодо аналізу, обговорення тематичного матеріалу на лекціях і лабораторних заняттях, ґрунтовної підготовки до занять, відсутності пропусків без поважних причин, виявлення поглиблених знань під час захисту звітів з лабораторного практикуму і модульного контролю.

2. МОВА ГІПЕРТЕКСТОВОЇ РОЗМІТКИ HTML5

2.1. Вступ в HTML5

2.1.1. Що таке HTML

HTML (HyperText Markup Language) – мова розмітки гіпертексту, що використовується для створення веб-сторінок. Саме тому веб-сторінки ще називають HTML-документами.

HTML почав свій шлях на початку 90-х років як примітивна мова створення веб-сторінок, а у **2014 році** офіційно була завершена робота над новим стандартом – **HTML5**, який фактично створив революцію, додаючи в HTML багато нового. Що саме приніс стандарт HTML5? Це, зокрема, новий алгоритм парсингу для створення структури DOM; додавання нових елементів і тегів, напр. елементів *video*, *audio* та ряд інших; перевизначення правил і семантики вже існуючих елементів HTML.

Із додаванням нових функцій HTML5 став не просто новою версією мови розмітки для створення веб-сторінок, а й фактично платформою для створення додатків. Причому сфера його використання вийшла далеко за межі веб-середовища інтернет: HTML5 застосовується також для створення мобільних додатків під Android, iOS, Windows Mobile і навіть десктопних додатків для персональних комп'ютерів (зокрема, в ОС Windows 8/8.1/10).

Хто відповідає за розвиток HTML5? Цим займається **World Wide Web Consortium** (скорочено, **W3C** – Консорціум Всесвітньої Павутини) – незалежна міжнародна організація, яка визначає стандарт HTML5 у вигляді специфікацій. Поточну повну специфікацію англійською мовою можна переглянути за адресою https://www.w3.org/TR/html5/. Зазначимо, що W3C продовжує працювати над HTML5, випускаючи оновлення до специфікації.

2.1.2. Елементи й атрибути HTML5

Розглянемо основні складові веб-сторінки.

HTML-документ складається **з елементів**, а елементи – **з тегів**. Як правило, елемент має відкриваючий та закриваючий тег, які беруться в кутові дужки. Наприклад,

* текст елемента div *

Вище визначено елемент *div*, який має відкриваючий тег * * і закриваючий тег * *, між цими тегами розташовується вмістиме елемента *div*. У даному випадку, це простий текст *“текст елемента div”*.

Назви тегів та елементів – це повноцінні англ. слова, їх перші літери або абревіатури. При написанні тегів **регістр літер значення не має**.

Елементи, що складаються з одного тега, називаються **одиничними тегами**. Наприклад, елемент *
*, призначення якого – перехід на новий рядок:

* Текст
 елемента div *

Такі елементи ще називають **порожніми** *(void elements)*. Хоча тут використовується закриваючий слеш, але його наявність, згідно зі специфікацією, необов’язкова і рівнозначна використанню тега без слеша: *
*.

Кожен елемент, всередині відкриваючого тега, може мати **атрибути**. Наприклад:

* Кнопка *

Елемент *div* має атрибут *style*, значення якого *“color:red;”* зазначається в лапках після знаку “дорівнює” і задає колір тексту, у даному випадку – червоний.

Існують ще **булеві, або логічні, атрибути** (*boolean attributes*) – це атрибути, які можуть не мати значення. Наприклад, для кнопки можна задати атрибут *disabled*, який вказує, що даний елемент відключений:

<input type="button" value="Натиснути" disabled>

Розрізняють **глобальні атрибути**, якими характеризуються всі елементи, і **специфічні**, використовувані тільки для певних елементів.

В HTML5 є **набір глобальних атрибутів**, застосовних до будь-якого елементу HTML5:

- ***accesskey*** визначає клавішу для швидкого доступу до елемента;
- ***class*** задає клас CSS, який буде застосований до елемента;
- ***contenteditable*** визначає, чи можна редагувати вміст елемента;

• ***contextmenu*** визначає контекстне меню для елемента, яке відображається при натисканні на елемент правою кнопкою миші;

• ***dir*** встановлює напрямок тексту в елементі;

• ***draggable*** визначає, чи можна переміщати елемент;

• ***dropzone*** визначає, що робити з перетягуваними користувачем даними – скопіювати, перетягнути або зв'язати – в момент, коли дані кидаються на елемент;

• ***hidden*** приховує елемент;

• ***id*** унікальний ідентифікатор елемента, на веб-сторінці елементи не повинні мати однакових ідентифікаторів;

• ***lang*** визначає мову елемента;

• ***spellcheck*** вказує, чи буде для цього елемента використовуватися перевірка правопису;

• ***style*** задає стиль елемента;

• ***tabindex*** визначає порядок, в якому буде виконуватися обхід елементів за допомогою клавіші *TAB*;

• ***title*** встановлює додатковий опис для елемента;

• ***translate*** визначає, чи буде перекладатися вмістиме елемента.

Але, як правило, з усього цього списку найбільш часто використовувані три: *class*, *id* і *style*.

В HTML5 було додано **користувацькі атрибути** (*custom attributes*). Користувацький атрибут – це атрибут, який визначений самим розробником, позначається префіксом *data-*. Наприклад:

<input type="button" value="Натиснути" data-color="red" >

Тут визначено атрибут *data-color*, який має значення "*red*". Хоча для цього елемента в HTML не існує такого атрибуту. Ми його визначаємо самі і встановлюємо для нього значення.

Часто в HTML при визначенні значень атрибутів застосовуються як **одинарні, так і подвійні лапки**. Наприклад:

<input type='button' value='Натиснути'>

<input type="button" value= 'Кнопка "Привіт світ"'>

І одинарні, і подвійні лапки в даному випадку допустимі, хоча частіше застосовуються саме подвійні лапки. Однак іноді саме значення атрибуту може містити подвійні лапки, тоді все

значення краще помістити в одинарні.

В HTML – документах можна використовувати **коментарі**, браузер ігнорує їх вміст, і не відображає їх на екрані. Коментар задається наступним чином:

<!– – вміст коментаря – –>

2.1.3. Структура HTML-сторінки та першого документа HTML5

Для створення html-сторінки потрібно створити звичайний текстовий файл, розширення якого змінити на **.html* або **.htm*.

Створимо текстовий файл, назвемо його *index* і змінимо його розширення на *.html*. Відкриваємо цей файл у будь-якому текстовому редакторі. Додамо у файл *index.html* наступний вміст:

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Документ HTML5</title>
   </head>
   <body>
      <div>Вмістиме документа HTML5</div>
   </body>
</html>
```

Документ HTML5 складається з HTML-елементів. Для створення документа HTML5 потрібні, в першу чергу, два елементи: *DOCTYPE* і *html*.

1. Елемент *DOCTYPE* (Document Type Declaration) повідомляє веб-браузеру тип документа. *<!DOCTYPE html>* показує, що даний документ є документом *html* і що використовується *html5*, а не *html4* або інша версія мови розмітки. Якщо вказати *DOCTYPE* неправильно або не вказати взагалі, то браузер покаже сторінку сумісну зі своїми старими версіями.

Формально **регістр не має значення**, але коректніше писати великими літерами, оскільки в XHTML5 є різниця між іменами елементів у нижньому регістрі і ключовими словами, записаними у верхньому регістрі.

2. Елемент *html* між своїм відкриваючим і закриваючим тегами містить вміст документа. Всередині тега *html* розміщуються два елементи: *head* і *body*.

3. Елемент *head* містить метадані веб-сторінки: її заголовок, тип кодування і т.д., а також посилання на зовнішні ресурси – стилі, скрипти, якщо вони використовуються. В елементі *head* визначено два вкладені теги:

• **елемент *title*** містить заголовок сторінки, який відображатиметься як заголовок вкладки браузера;

• **елемент *meta*** визначає метаінформацію сторінки. Для коректного відображення символів бажано вказувати кодування. У даному випадку за допомогою атрибуту *charset="utf-8"* вказано кодування *utf-8*, тому браузер коректно відображатиме кириличні символи.

4. Елемент *body* містить вміст *html*-сторінки. Весь текст, визначений усередині елемента *body*, ми побачимо в основному вікні браузера.

Оскільки ми вибрали кодування **utf-8**, то браузер буде відображати веб-сторінку саме в цьому кодуванні. Однак необхідно, щоб сам текст документа також відповідав вказаному кодуванню utf-8. Як правило, в текстових редакторах є відповідні налаштування для встановлення кодування. Наприклад, в Notepad++ потрібно зайти в меню *Кодування* і в переліку вибрати пункт *Перетворити в UTF-8 без BOM*.

Після цього в рядку стану можна буде побачити *UTF-8 w/o BOM*. Це показує, що вибрано потрібне кодування.

Збережемо і відкриємо файл *index.html* у браузері:

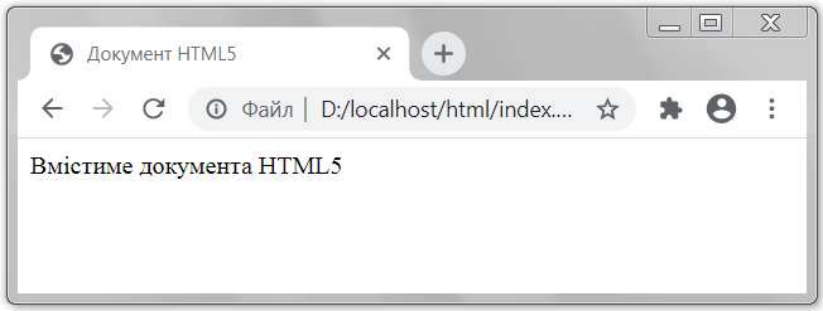


Рис. 2.1. Перший документ HTML5

Таким чином, ми створили перший документ HTML5.

2.1.4. Різновиди синтаксису HTML5

При створенні документа HTML5 можна використовувати два різні стилі: HTML і XML.

Стиль HTML передбачає наступні плавила:

1. Відкриваючі теги можуть бути відсутніми в елементів.
2. Закриваючі теги можуть бути відсутніми в елементів.
3. Тільки порожні елементи (*void elements*) (наприклад, *br*, *img*, *link*) можуть закриватися за допомогою слеша />.
4. **Регістр назв тегів і атрибутів** не має значення.
5. **Значення атрибутів** можна не брати у лапки.
6. Деякі атрибути можуть не мати значень (напр., *checked* і *disabled*).
7. Спеціальні символи НЕ екрануються (екранування символів – заміна в тексті управляючих символів на відповідні текстові підстановки).
8. Документ повинен мати елемент *DOCTYPE*.

Це так званий "дозвільний" стиль, що ґрунтується на послабленнях при створенні документу.

Документ HTML5 також може бути описаний за допомогою синтаксису **XML**. Такий стиль ще називають **XHTML** (eXtensible HyperText Markup Language) – розширена мова розмітки гіпертексту. XHTML багато в чому ідентичний HTML, але більш строгий, ніж HTML. Він використовується, якщо заголовок ***content-type*** має значення ***application/xml+xhtml***. Для цього стилю характерні наступні правила:

1. Кожен елемент повинен мати відкриваючий тег.
2. Непорожні елементи *(non-void elements)* із відкриваючим тегом також повинні мати й закриваючий тег.
3. Будь-який елемент може закриватися за допомогою слеша />.
4. Назви тегів і атрибутів, **чутливі до регістру**, як правило, записуються в нижньому регістрі.
5. **Значення атрибутів** повинні зазначатися в лапках.
6. Атрибути без значень не допускаються (*checked="checked"* замість просто *checked*).
7. Спеціальні символи повинні бути екрановані.

Порівняємо два підходи. **Підхід HTML5**:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8>
    <title> Документ HTML5</title>
  </head>
  <body>
    <p>Вмістиме документа <br>
    <input type=button value= Натиснути >
  </body>
</html>
```

Аналогічний приклад із використанням **підходу XHTML**:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta charset="utf-8">
    <title> Документ XHTML </title>
  </head>
  <body>
   <p> Вмістиме документа <br/>
     <input type="button" value="Натиснути" />
   </p>
  </body>
</html>
```

При використанні синтаксису **XHTML** потрібно вказати простір імен для даного документа:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

XHTML підтримується всіма основними браузерами.

Вибір конкретного стилю при написанні html-документів залежить від побажань програміста або веб-дизайнера. Також використовується **змішаний стиль**, який запозичує правила одночасно з першого і з другого стилів.

В той же час потрібно враховувати, що наявність в елемента закриваючого і відкриваючого тегів зменшує ймовірність того, що елемент буде неправильно інтерпретований браузером.

Також узяття значень атрибутів в лапки допомагає уникну-

ти потенційних помилок. Так, атрибут *class* може приймати кілька значень, розділених пробілом, наприклад: `<div class="navmenu bigdesctop">`. Якщо опустити лапки, то значенням атрибуту буде *"navmenu"*, а *"bigdesctop"* браузер буде намагатися інтерпретувати як окремий атрибут.

Якщо виникають питання, наскільки правильна створювана розмітка html, то її можна перевірити за допомогою валідатора за адресою https://validator.w3.org.

2.2. Елементи в HTML5

Основна частина документа html, фактично все, що бачимо у браузері при завантаженні веб-сторінки, розташовується між відкриваючим та закриваючим тегами елемента *body*: `<body>…</body>`, де відповідно розміщується більшість елементів html.

Хоча більшість елементів HTML5 залишаються тими самими, що і в попередніх версіях, але дещо змінився спосіб їх використання. Розглянемо базові елементи HTML5, їх призначення та використання.

2.2.1. Елемент meta та метадані веб-сторінки

Крім задання кодування за допомогою атрибуту *charset*, елемент *meta* також має два атрибути: *name* і *content*. Атрибут *name* містить назву метаданих, а *content* – їх значення.

За замовчуванням, в HTML визначені **п'ять типів метаданих**:

- ***application name***: назва веб-додатку, частиною якого є даний документ;
- ***author***: автор документа;
- ***description***: короткий опис документа;
- ***generator***: назва програми, яка згенерувала даний документ;
- ***keywords***: ключові слова документа.

Зазначимо, що найбільш актуальний **тип *description***. Його значення пошукові системи часто використовують як анотацію документа у пошуковій видачі.

Додамо в документ набір елементів *meta*:

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <base href="content/">
    <title>Елемент title</title>
    <meta name="description"
          content="Перший документ HTML5">
    <meta name="author" content="Bill Gates">
 </head>
 <body>
 <a href="newpage.html">Вмістиме документа HTML5</a>
 </body>
</html>
```

Елемент *base* дозволяє вказати базову адресу, відносно якої встановлюються інші адреси, використовувані в документі:

```
<base href="content/">
```

Таким чином, хоча адресою посилання (атрибут *href*) вказано сторінку *newpage.html*, але фактично її адресою буде *content/newpage.html*, тобто в одній папці з поточною сторінкою повинна бути підпапка *content*, в якій має знаходитися файл *newpage.html*.

Можна також вказувати повну адресу:

```
<base href="http://www.microsoft.com/">
```

У цьому випадку посилання виконує перехід за адресою *http://www.microsoft.com/newpage.html*.

2.2.2. Елементи групування

Елементи для групування контенту на веб-сторінці:

1. Елемент *div* призначений для структуризації контенту на веб-сторінці, а саме розташування наповнення сторінки в окремих блоках. *div* створює блок, який, за замовчуванням, розтягується по всій ширині браузера, а наступний після *div* елемент переноситься на новий рядок.

```
<div>Вмістиме елемента div</div>
```

2. Параграфи створюються за допомогою тега **<*p*>…</*p*>**, який містить наповнення. Кожен новий параграф розташовується з нового рядка і фактично визначає новий абзац.

Якщо зустрівся цей тег, браузер не тільки перейде на новий рядок, щоб почати новий абзац, але і виведе попереду цього абзацу порожній рядок.

Якщо в рамках одного параграфа потрібно перенести текст на новий рядок, то можна використати елемент **<*br*>** – тег переходу на новий рядок:

* Перший рядок.
 Другий рядок. *

3. Елемент <*pre*> виводить блок попередньо відформатованого тексту. Такий текст відображається моноширинним шрифтом і з усіма пробілами між словами. За замовчуванням, будь-яка кількість пробілів підряд на веб-сторінці відображається як один пробіл. Тег *<pre>* дозволяє обійти цю особливість і відображати текст, як потрібно розробнику.

4. Елемент span призначений переважно для стилізації вкладеного в нього тексту, а саме дозволяє виділити частину інформації всередині інших тегів і встановити для неї свій стиль. На відміну від блоків *div* або параграфів, *span* не переносить вміст на наступний рядок.

Використаємо перераховані елементи:

```
<!DOCTYPE html>
<html>
 <head>
     <meta charset="utf-8">
     <title>Документ HTML5</title>
 </head>
 <body>
 <div>Заголовок документа HTML5</div>
 <div>
  <p><span style="color:red; text-decoration:underline;">
         Перший </span>параграф</p>
  <p><span>Другий</span> параграф</p>
  <p>Перший абзац<br/> Другий абзац</p>
  <pre>
            Перший рядок
            Другий рядок
            Третій рядок
  </pre>
 </div>
```

```
  </body>
</html>
```

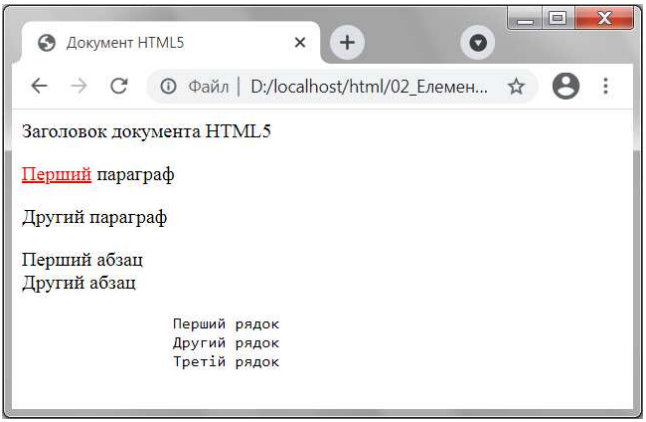


Рис. 2.2. Елементи групування

Сам по собі *span* нічого не робить. Зокрема, у другому параграфі *span* жодним чином не вплинув на текст, у ньому розташований. А в першому параграфі елемент *span* містить атрибут стилю: *style="color:red; text-decoration:underline;"*, який визначає для вкладеного тексту червоний колір шрифту та підкреслення.

Елементи ***div* і *p* блокові**, елемент *div* може містити будь-які інші елементи, а елемент *p* – тільки рядкові. На відміну від них, елемент ***span* є рядковим**, тобто ніби вбудовує свій вміст у зовнішній контейнер – той же *div* або параграф. Але при цьому не слід поміщати блокові елементи у рядковий елемент *span*.

2.2.3. Заголовки та форматування тексту

Елементи <*h1*>, <*h2*>, <*h3*>, <*h4*>, <*h5*> і <*h6*> призначені для створення **заголовків різного рівня**. Заголовки задають шрифт жирним і, за замовчуванням, мають деякий розмір: від найбільшого <*h1*> до найменшого <*h6*>. При визначенні заголовків потрібно врахувати, що на сторінці повинен бути тільки один заголовок першого рівня, тобто <*h1*>, який виконує роль основного заголовку веб-сторінки.

Наприклад, *<hn>Заголовок n-рівня</hn>*, де *n* – номер заголовку відповідного рівня.

В HTML розмір шрифту вимірюється в умовних одиницях від 1 до 7, розмір тексту, що використовується за замовчуванням, становить 3.

Розглянемо набір елементів HTML, призначених для **форматування текстового наповнення:**

- **<*b*>** виділяє текст жирним;
- **<*i*>** виділяє текст курсивом;
- **<*u*>** підкреслює текст;
- **<*s*>** відображає текст як закреслений;
- **<*sub*>** текст у вигляді підрядкового тексту (нижній індекс), напр. x_2;
- **<*sup*>** текст у вигляді надрядкового тексту (верхній індекс), напр. x^2;
- **<*small*>** зменшує розмір шрифту на одиницю у порівнянні із сусіднім текстом. Допускається застосування вкладених тегів *<small>*, при цьому розмір шрифту буде зменшуватися з кожним рівнем вкладеності, але не може бути менше ніж 1.

Застосуємо ці елементи:

```
<!DOCTYPE html>
<html>
 <head>
     <meta charset="utf-8">
     <title>Заголовки та форматування тексту
          в HTML5</title>
 </head>
 <body>
 <h2>Форматування тексту в HTML5</h2>
     тег b: текст <b> жирним </b> шрифтом
 <br>тег i: текст <i>курсивом</i>
 <br>тег u: <u>підкреслений </u> текст
 <br>тег s: <s> закреслений </s> текст
 <br>тег sub: нижній індекс: X<sub>i</sub>
         
     тег sup: верхній індекс: Y<sup>2</sup>
 <br>тег small: <small>зменшений текст</small>
```

</body>
</html>

Результуюча сторінка має вигляд:

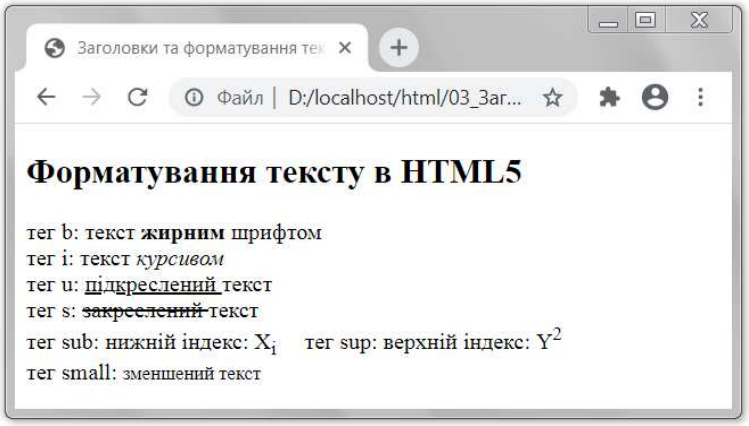


Рис. 2.3. Форматування тексту в HTML5

2.2.4. Робота із зображеннями

В HTML для виведення зображень використовується **елемент *img***, який має два основні атрибути:

1. Обов'язковий атрибут *src* задає місцезнаходження зображення: якщо файл знаходиться в мережі, потрібно вказати повну URL-адресу; якщо файл знаходиться на локальному комп'ютері, то достатньо вказати шлях до нього (від поточного або кореневого каталога), напр., *<img src="pic.gif">*. Якщо шлях до файлу не вказано, то виконується пошук у поточному каталозі.

2. Необов'язковий атрибут *alt* визначає текстовий опис зображення. Якщо браузер з якихось причин не може відобразити зображення (наприклад, якщо в атрибуті *src* некоректно задано шлях), то браузер замість картинки показує текстовий опис.

Наприклад, щоб відобразити зображення на веб-сторінці, розмістимо його в тій самій папці, що й веб-сторінка, та використаємо тег *img*:

**

Використовуючи стильові особливості, зокрема відступи і обтікання, можна комбінувати зображення з текстом. Наприклад:

```
<img src="LoremIpsum.jpg" alt="Lorem Ipsum"
     style="float:left; margin-right:10px;" />
<h1>Lorem Ipsum</h1>
<b>Lorem Ipsum</b> dolor sit amet, consectetur adipiscing…
```

Рис. 2.4. Зображення на веб-сторінці

2.2.5. Списки

Мова HTML підтримує створення кількох типів списку, зокрема ненумерованого (маркованого) та нумерованого.

Для створення списків в HTML5 застосовуються елементи *<ol>* (нумерований список) і *<ul>* (маркований список). Для задання елементів списку використовується елемент *<li>*.

Наприклад,

```
<h3>Нумерований список</h3>
<ol>
  <li>Samsung Galaxy A50</li>
  <li>Xiaomi Mi Note 10</li>
  <li>Huawei P Smart</li>
</ol>
<h3>Ненумерований список</h3>
```

```
<ul>
  <li>Samsung Galaxy A50</li>
  <li>Xiaomi Mi Note 10</li>
  <li>Huawei P Smart</li>
</ul>
```

Ці списки на сторінці мають вигляд:

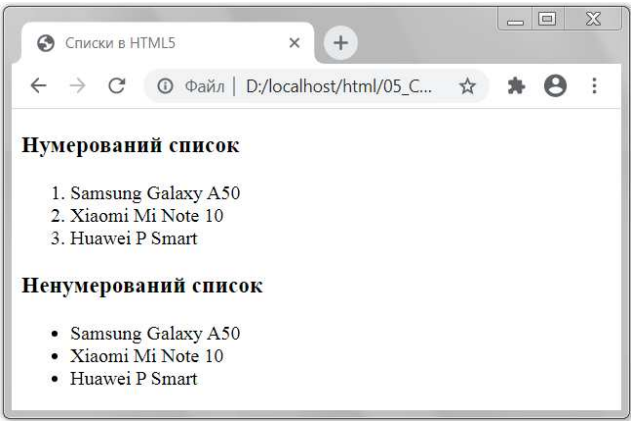


Рис. 2.5. Створення списків

За замовчуванням, у нумерованому списку для нумерації елементів використовується стандартні цифри від 1, у маркованому – кожен елемент позначається чорною крапкою.

При необхідності можна налаштувати нумерацію або відображуваний, поряд з елементом, символ за допомогою стилю *list-style-type*. Цей стиль може набувати багато різних значень. Зазначимо тільки основні і часто використовувані.

Для нумерованих списків стиль *list-style-type* може набувати таких значень:

- *decimal*: десяткові числа, відлік з 1;
- *decimal-leading-zero*: десяткові числа, в яких одинарні цифри записуються з нулем попереду, наприклад, 01, 02, 03, ..., 97, 98, 99;
- *lower-roman*: малі римські цифри, наприклад, i, ii, iii, iv, ;
- *upper-roman*: великі римські цифри, наприклад, I, II, III,...;
- *lower-alpha*: малі латинські літери, наприклад, a, b, c, ..., z;
- *upper-alpha*: великі латинські літери, напр.., A, B, C,... Z.

Для нумерованих списків за допомогою **атрибуту *start*** можна додатково задати символ, з якого буде починатися нумерація. Наприклад:

```
<h3>Нумерований список list-style-type:decimal </h3>
<ol style="list-style-type:decimal;" start="3">
  <li>Samsung Galaxy A50</li>
  <li>Xiaomi Mi Note 10</li>
  <li>Huawei P Smart</li>
</ol>
```

Для маркованого списку атрибут *list-style-type* може набувати таких значень: *disc*: чорний диск, *circle*: порожній кружечок, *square*: чорний квадратик. Наприклад,

```
<h3>Маркований список list-style-type: square</h3>
<ul style="list-style-type:square;">
  <li>Samsung Galaxy Note 10 Plus</li>
  <li>Samsung Galaxy S10+</li>
  <li>Samsung Galaxy S10e</li>
</ul>
```

Останні два списки, на сторінці виглядають наступним чином:

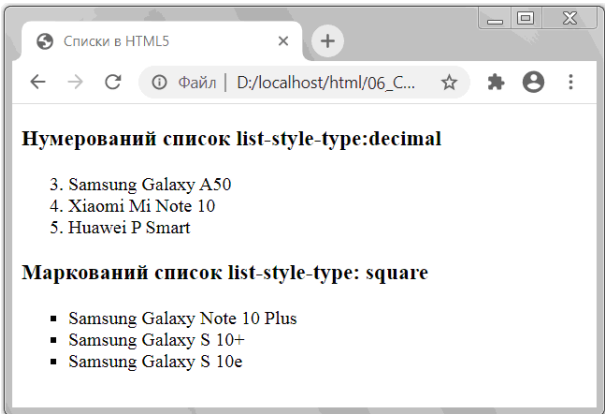


Рис. 2.6. Створення списків

Ще одну цікаву можливість із налаштування списків надає **стиль *list-style-image***, а саме використання зображення як маркера списку:

*<ul style="list-style-image:url(phone_touch.png);">…
*

Стиль *list-style-image* має значення *url(phone_touch.png)*, де *“phone_touch.png”* – назва файлу зображення, що знаходиться в одній папці з веб-сторінкою.

2.2.6. Елемент details

Елемент *details* використовується для зберігання інформації, яку можна приховати або показати за вимогою користувача. Тег *details* створює **блок, що розкривається**, який, за замовчуванням, відображається у згорнутому вигляді. Якщо встановити **атрибут *open***, то, за замовчуванням, блок відображатиметься у розгорнутому вигляді.

Тег *details* містить **елемент *summary***, який представляє заголовок блоку, і цей заголовок відображається у прихованому режимі, а також контент, що відображається після клацання користувачем по заголовку блоку. Контент може бути будь-яким, напр. текст, що представляє інструкцію або список елементів.

Наприклад, заголовок *“Флагмани”* і список елементів, що відображатиметься після розкриття блоку *“Флагмани”*:

```
<details>
  <summary>Флагмани 2020</summary>
  <ul>
     <li>Huawei P40 Pro</li>
     <li>Honor 30 Pro+ </li>
     <li>Samsung Galaxy Note 20 Ultra</li>
  </ul>
</details>
```

На сторінці виглядає наступним чином:

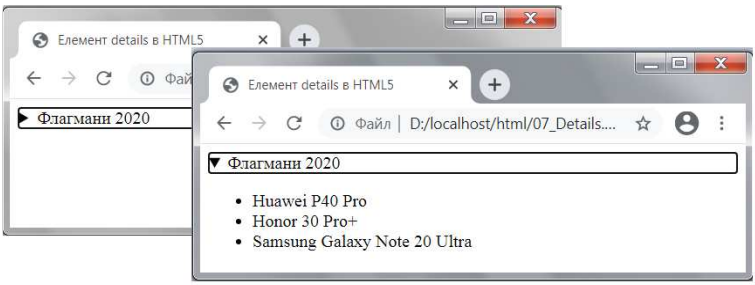


Рис. 2.7. Елемент *details*

За замовчуванням, ми бачимо тільки заголовок *summary*, натиснувши на стрілку або заголовок, можна розкрити блок.

2.2.7. Список визначень

Список визначень (англ. *definition list*) являє собою список, що містить набір пар “термін-визначення”, тобто кожен елемент списку складається з терміну й визначення.

Для створення такого списку використовується **тег <*dl*>…</*dl*>**. Всередину цього тегу поміщаються елементи списку: **термін** поміщається в тег **<*dt*>…</*dt*>** (*dt* скор. від "*definition term*"), а **визначення** – у тег **<*dd*>…</*dd*>** (*dd* скор. від "*definition description*").

У загальному випадку це виглядає так:

<*dl*>
<*dt*>*Термін1***</*dt*>**
<*dd*>*Визначення терміну1***</*dd*>**
<*dt*>*Термін2***</*dt*>**
<*dd*>*Визначення терміну2***</*dd*>**
</*dl*>

У тегів **<*dl*>**, **<*dt*>**, **<*dd*>** відсутні власні атрибути. Тег <*dd*> може містити абзаци, розриви рядків, зображення, посилання, списки і т.д.

Приклад. Розглянемо найпростіший список визначень:

```
<dl>
  <dt>HTML </dt>
  <dd>мова розмітки гіпертексту, що використовується
      для створення веб-сторінок. </dd>
  <dt>CSS</dt>
  <dd>набір параметрів форматування, що
      застосовуються до елементів документа … .</dd>
  <dt>JavaScript</dt>
  <dd>мова програмування, що найбільш широко
      використовується для надання інтерактивності
      веб-стоірнккам. </dd>
</dl>
```

Візуально такий список визначень має вигляд:

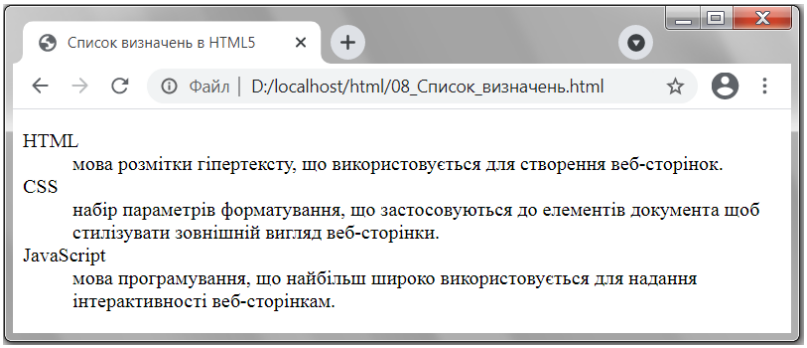


Рис. 2.8. Список визначень у HTML5

2.2.8. Таблиці

Для створення таблиць в HTML використовується **елемент *table***. Таблиця (*<table> … </table>*) містить набір рядків, кожний з яких описується **елементом *tr*** (*<tr>…</tr>*), а кожен рядок складається з комірок у вигляді **елементів *td*** (*<td>…</td>*). Кількість комірок у різних рядках таблиці необов'язково повинна бути однаковою. При цьому рядки з меншою кількістю комірок будуть автоматично доповнені справа порожніми комірками.

Атрибути *colspan* і *rowspan* елемента *td* використовуються для **об'єднання комірок**: ***colspan*** задає кількість стовпців, які охоплює по горизонталі дана комірка, ***rowspan*** задає кількість рядків, які вона охоплює по вертикалі. За замовчуванням, ці атрибути мають значення 1. Рядки, на які планується розширити дану комірку повинні існувати, інакше нічого не відбудеться.

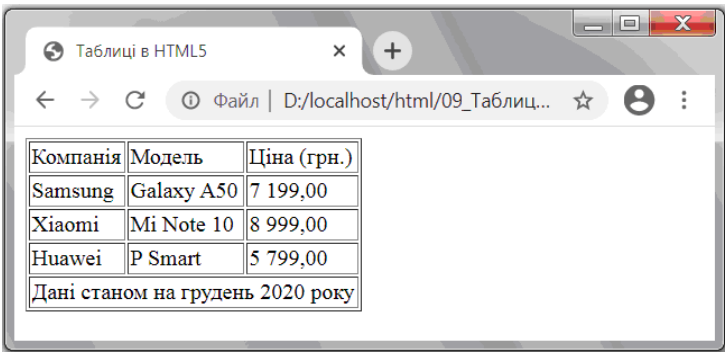
Щоб задати **товщину рамки таблиці** у пікселях використовується **атрибут *border*** елемента *table*. Якщо цей атрибут відсутній, або його значення дорівнює нулю, то рамки не виводяться (рамка таблиці невидима).

Приклад. Створимо найпростішу таблицю:

```
<table border=1>
  <tr>
 <td>Компанія</td><td>Модель</td><td>Ціна (грн.)</td>
  </tr>
  <tr>
 <td>Samsung</td><td>Galaxy A50</td><td>7 199,00</td>
```

```
  </tr>
  <tr>
 <td>Xiaomi</td><td>Mi Note 10</td><td>8 999,00</td>
  </tr>
  <tr>
 <td>Huawei</td><td>P Smart</td><td>5 799,00</td>
  </tr>
  <tr>
 <td colspan="3">Дані станом на грудень 2020 року</td>
  </tr>
</table>
```

В описаній таблиці п’ять рядків і три стовпці. При цьому перший рядок виконує роль заголовка, наступні три містять дані, а в останньому рядку перша комірка об’єднана з двома наступними. Така таблиця має вигляд:



Компанія	Модель	Ціна (грн.)
Samsung	Galaxy A50	7 199,00
Xiaomi	Mi Note 10	8 999,00
Huawei	P Smart	5 799,00
Дані станом на грудень 2020 року		

Рис. 2.9. Найпростіша таблиця в HTML

Для розділення заголовків, футера і тіла таблиці в HTML передбачені, відповідно, елементи *thead*, *tfoot* і *tbody*.

Елемент *thead* містить рядок заголовків. Для комірок-заголовків використовується не елемент *td*, а *th*, який виділяє заголовок жирним. Всі інші рядки містяться в ***tbody***.

Елемент *tfoot* визначає підвал таблиці або футер, де зазвичай виводиться допоміжна інформація про таблицю.

Крім заголовків стовпців, за допомогою **елемента *caption*** можна задати заголовок для таблиці.

Використовуючи ці елементи, змінимо таблицю наступним чином:

```
<table border=1>
 <caption> Популярні смартфони 2020</caption>
<thead>
  <tr>
      <th>Компанія</th>
      <th>Модель</th>
      <th>Ціна (грн.)</th>
  </tr>
</thead>
<tbody>
  <tr>
      <td>Samsung</td>
      <td>Galaxy A50</td>
      <td>7 199,00</td>
  </tr>
  <tr>
      <td>Xiaomi</td>
      <td>Mi Note 10</td>
      <td>8 999,00</td>
  </tr>
  <tr>
       <td>Huawei</td>
       <td>P Smart</td>
       <td>5 799,00</td>
  </tr>
</tbody>
<tfoot>
  <tr>
   <th colspan="3">
       Дані станом на грудень 2020 року
   </th>
  </tr>
</tfoot>
</table>
```

Тепер таблиця має вигляд:

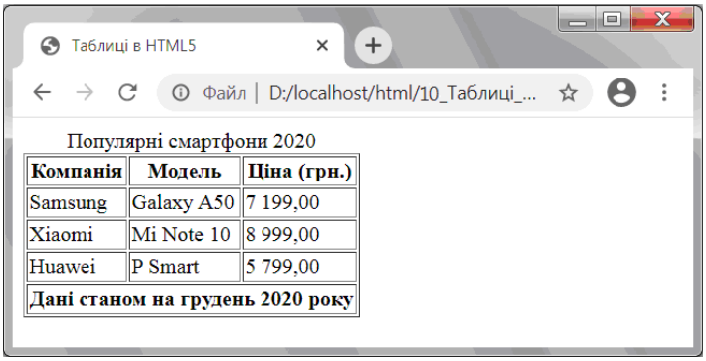


Рис. 2.10. Таблиця в HTML. Елементи *thead*, *tbody* і *tfoot*

2.2.9. Посилання

Посилання забезпечують навігацію між окремими документами. Посилання створюється за допомогою **елемента <*a*>…</*a*>**, який має такі **атрибути**:

- ***href*** визначає адресу посилання;
- ***hreflang***: мова документа, на який вказує дане посилання;
- ***media*** визначає пристрій, для якого призначене посилання;
- ***rel*** визначає відношення між даним документом і ресурсом за посиланням;
- ***target*** визначає, де саме буде відкриватися документ, на який вказує посилання;
- ***type*** вказує на mime-тип ресурсу за посиланням.

Наприклад,

```
<a href="content.html">Вивчення HTML5</a>
```

де для посилання використовується **відносний шлях** *content.html*. Це означає, що в одній папці із даним документом повинен знаходитися файл *content.html*, на який відбуватиметься перехід при натисканні на посилання.

Також можна використовувати **абсолютний шлях** із повним зазначенням адреси:

```
<a href="http://metanit.com/web/html5/">
    Вивчення HTML5
</a>
```

За замовчуванням, ресурси, на які ведуть посилання, відкриваються в тому ж вікні. За допомогою **атрибуту *target*** можна

перевизначити цю поведінку. Атрибут *target* може приймати **такі значення**:

- ***_blank***: відкриття html-документа в новій вкладці браузера;
- ***_self***: відкриття html-документа в тому ж фреймі (або вікні);
- ***_parent***: відкриття сторінки у батьківському фреймі, якщо посилання розташовано у внутрішньому фреймі;
- ***_top***: відкриття html-документа на все вікно браузера;
- ***framename***: відкриття html-документа у фреймі з назвою "*framename*" (де "*framename*" – довільна назва фрейму).

Наприклад, якщо потрібно відкрити документ за посиланням, у новій вкладці браузера використовують атрибут *target*:

```
<a href="http://metanit.com/web/html5/" target="_blank">
  HTML5
</a>
```

Для того щоб розмістити **внутрішні посилання** для реалізації переходів по певних блоках всередині елементів, потрібно вказати в атрибуті *href* знак решітки (#), після якого *id* того елемента, до якого треба здійснити перехід. Наприклад, перехід до заголовків *h2*:

```
<a href="#paragraph1">Параграф 1</a> |
<a href="#paragraph2">Параграф 2</a> |
<a href="#paragraph3">Параграф 3</a>
<h2 id="paragraph1">Параграф 1</h2>
<p>Вмістиме параграфа 1</p>
<h2 id="paragraph2">Параграф 2</h2>
<p> Вмістиме параграфа 2</p>
<h2 id="paragraph3">Параграф 3</h2>
<p> Вмістиме параграфа 3</p>
```

Розмістивши всередині елемента *<a>* елемент *<img>*, можна зробити зображення посиланням:

```
<a href="index.html">
  <img src="cover.png" alt="HTML">
</a>
```

2.2.10. Елементи figure і figcaption

Елемент *figure* використовується для групування будь-яких елементів, що ілюструють або підтримують певну ідею тексту (контенту). Елемент *figure* може містити зображення, відеоролик, схему, фрагмент коду, діаграму або навіть таблицю – майже все, що може з'явитися в потоці веб-контенту і має сприйматися як автономна одиниця. Тег *figure* не має власних атрибутів.

Елемент *figcaption* виводить заголовок або пояснення для вмісту всередині елемента *figure*.

Для використання елемента *figure* потрібно помістити в нього деякий вміст, у найпростішому випадку зображення.

Приклад використання елементів ***figure*** і ***figcaption***:

```
<body>
  <div>
    <p>Lorem ipsum dolor … </p>
      <figure>
        <figcaption>Березень 2021</figcaption>
        <img src="Lorem_Ipsum.jpg" alt="Lorem Ipsum">
      </figure>
     <p>What is Lorem Ipsum? </p>
  </div>
<body>
```

Сторінка має вигляд:

Рис. 2.11. Елементи *figure* і *figcaption* в HTML

2.2.11. Фрейми

Фрейми (англ. *frame*)) дозволяють завантажити в область заданих розмірів на веб-сторінці іншу веб-сторінку або ресурс. Фрейми створюються **елементом *iframe***. Наприклад, вбудуємо на веб-сторінку стартову сторінку вікіпедії:

```
<iframe src="http://wikipedia.com"
        width="400" height="200">
</iframe>
```

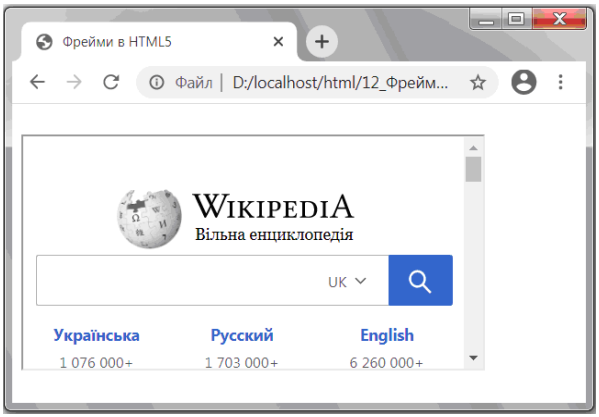


Рис. 2.12. Фрейми в HTML

Елемент *iframe* не містить у собі ніякого вмісту. Налаштування фрейму здійснюється за допомогою **атрибутів:**

- ***src*** містить повний шлях до завантажуваного ресурсу;
- ***width***: ширина фрейму;
- ***height***: висота фрейму.

Зазначимо, що не всі сайти можуть відкриватися у фреймах, оскільки на стороні веб-сервера можуть діяти обмеження на відкриття у фреймах.

2.3. Робота з формами

2.3.1. Форми

Форми в HTML використовуються для вводу і відправки даних користувачем. Форма на *html*-сторінці описується тегом ***<form>…</form>***, всередині якого, відповідно, розміщуються всі поля форми.

Для налаштування форми **елемент *form* має такі атрибути**:

- **method**: метод відправки даних форми на сервер, можливі два значення: *post* і *get*; **значення *post*** дозволяє передати дані на веб-сервер через спеціальні заголовки, а **значення *get*** дозволяє передати дані через рядок запиту;
- **action** задає адресу, на яку передаються дані форми;
- **enctype** визначає спосіб кодування даних форми при передачі їх на сервер (використовується тільки при *method="post"*); зазвичай встановлювати значення атрибуту *enctype* не потрібно, дані цілком правильно розуміються на стороні сервера. Однак якщо використовується поле для відправки файлу (<*input type="file"*>), слід визначити атрибут *enctype* як *multipart/form-data.*

Атрибут **enctype** може набувати таких значень:

o ***application/x-www-form-urlencoded***: кодування даних, що передаються, за замовчуванням (замість пробілів проставляється +, символи кодуються шістнадцятковими значеннями);

o ***multipart/form-data****:* кодування, що використовується при відправці файлів;

o ***text/plain***: використовується при відправці простої текстової інформації (пропуски замінюються знаком +, букви та інші символи не кодуються).

Наприклад:

```
<form method="post"
        action="http://localhost:8080/login.php">
</form>
```

Для форми встановлено метод "*post*", тобто всі значення форми відправляються в тілі запиту, а адресою відправлення є рядок *http://localhost:8080/login.php*. Адресу вказано довільним чином.

Як правило, за вказаною адресою працює веб-сервер, який, використовуючи одну з технологій серверної сторони (PHP, NodeJS, ASP.NET і т.д.), може отримувати запити й повертати відповідь.

Автозаповнення

Часто веб-браузери запам'ятовують дані, що вводяться, і при введенні у відповідні поля форми, можуть видавати список підказок із раніше введених слів. Це може бути не завжди бажано, і за допомогою **атрибуту *autocomplete*** зі значенням *"off"* можна відключити автозаповнення форми. Якщо потрібно включити автозаповнення тільки для якихось певних полів, то можна застосувати до них атрибут *autocomplete="on"*:

```
<form method="post" autocomplete="off"
        action="http://localhost:8080/login.php">
    <input name="login" autocomplete="on"/>
    <input name="password" />
    <input type="submit" value="Увійти" />
</form>
```

Тепер для всієї форми, крім першого поля, буде відключено автозаповнення:

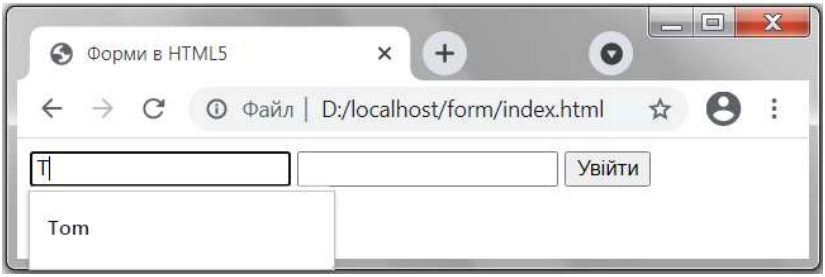


Рис. 2.13. Форма в HTML5

Автозаповнення може бути відключено у налаштуваннях браузера, в такому випадку атрибут *autocomplete* працювати не буде.

2.3.2. Елементи форми

Всі елементи, які потрібно розташувати на формі, розміщуються у тезі ***<form>… </form>***. Найбільш поширеним елементом вводу є **елемент *input***. Однак реальна дія цього елемента залежить від того, яке значення встановлено для його **атрибуту *type***, який може набувати таких значень:

- ***text***: звичайне текстове поле;
- ***password***: також текстове поле, але замість символів, що вводяться, відображатимуться зірочки, тому використовується переважно для вводу паролю;
- ***radio***: радіокнопка або перемикач (з групи радіокнопок можна вибрати тільки одну);
- ***checkbox***: елемент прапорець, який може знаходитися у відміченому або невідміченому стані;
- ***hidden***: приховане поле;
- ***submit***: кнопка відправки даних з форми;
- ***color***: поле для вводу кольору;
- ***date***: поле для вводу дати;
- ***datetime***: поле для вводу дати і часу з урахуванням часового поясу;
- ***datetime-local***: поле для вводу дати і часу без врахування часового поясу;
- ***email***: поле для вводу адреси електронної пошти;
- ***month***: поле для вводу року і місяця;
- ***number***: поле для вводу чисел;
- ***range***: повзунок для вибору числа з деякого діапазону;
- ***search***: поле пошуку, яке на формі виглядає як просте текстове поле;
- ***tel***: поле для вводу телефону;
- ***time***: поле для вводу часу;
- ***week***: поле для вводу року і тижня;
- ***url***: поле для вводу адреси url;
- ***file***: поле для вибору файлу для відправки;
- ***image***: створює кнопку у вигляді картинки.

Крім елемента *input* у різних модифікаціях, є ще **невеликий набір елементів, які також можна використовувати на формі**:

- ***button***: створює кнопку;

- ***select***: випадаючий список;
- ***label***: створює надпис, який відображається поруч з полем вводу;
- ***textarea***: багаторядкове текстове поле.

Атрибути *name* і *value*

Для всіх елементів вводу можна задати атрибути *name* і *value*. **Атрибут *name*** дозволяє ідентифікувати поле вводу, а **атрибут *value*** задає значення поля вводу. Наприклад:

```
<form method="get" action="index.html">
    <input type="text" name="login" value="Tom"/>
    <input type="password" name="password"/>
    <input type="submit" value="Увійти" />
</form>
```

Таким чином, при завантаженні веб-сторінки текстове поле вже містить текст *"Tom"* (як зазначено в атрибуті *value* цього поля):

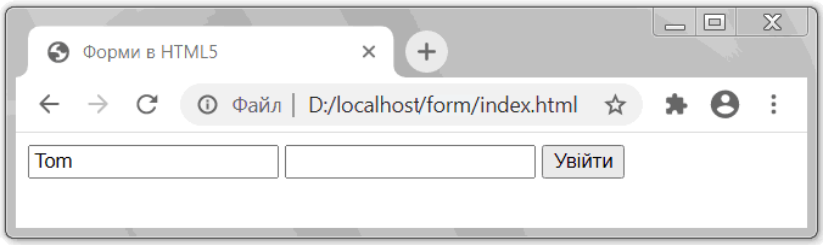


Рис. 2.14. Форма з полями для вводу імені та паролю

Методом відправки даних форми є **метод *"get"***, тому дані будуть відправлятися через рядок запиту. В цьому випадку не важливо, як дані будуть прийматися на сервері, не важливий сервер, який їх отримуватиме. Тому адресою вказано ту саму сторінку, тобто файл *index.html*. І при відправці ми можемо побачити введені дані в рядку запиту:

*file:///D:/localhost/form/index.html?**login=Tom&password=qwerty***

При відправці даних з форми браузер з'єднує всі дані у набір пар "ключ – значення". У нашому випадку дві таких пари: *login=Tom* і *password=qwerty*. Ключем в цих парах виступає назва поля вводу, яка визначається атрибутом *name*, а значенням –

те значення, яке введено в поле вводу (або значення атрибуту *value*).

Отримавши ці дані, сервер легко може дізнатися, які значення в які поля вводу були введені користувачем.

2.3.3. Кнопки

Кнопки, представлені **елементом *button***, мають широкі можливості по конфігурації. Так, в залежності від **значення атрибуту *type*** можна створити різні типи кнопок:

- ***submit***: кнопка відправки даних форми;
- ***reset***: кнопка скидання значень форми;
- ***button***: кнопка без спеціального призначення.

Якщо **кнопка використовується для відправки даних форми**, тобто встановлено атрибут ***type="submit"***, то можна задати для неї ряд додаткових атрибутів:

- ***form***: форма, на якій розміщена кнопка відправки;
- ***formaction***: адреса, на яку відправляються дані з форми; якщо у елемента *form* заданий атрибут *action*, то він перевизначається;
- ***formenctype***: встановлює формат відправки даних; якщо у елемента *form* встановлений атрибут *enctype*, то він перевизначається;
- ***formmethod***: встановлює метод відправки даних з форми (***post*** **або** ***get***). Якщо в елемента *form* встановлено атрибут *method*, то він перевизначається.

Наприклад, визначимо на формі кнопку відправки і кнопку скидання (очищення від введених даних):

```
<form>
  <p><input type="text" name="login"/></p>
  <p><input type="password" name="password"/></p>
  <p>
  <button type="submit" formmethod="get"
          formaction="index.html"> Відправити
  </button>
  <button type="reset">Очистити</button>
  </p>
</form>
```

Створена форма має вигляд:

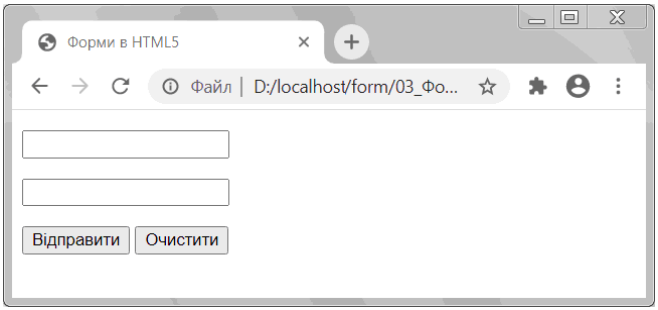


Рис. 2.15. Кнопки на формі

Крім елемента *button*, для створення кнопок на формі можна використовувати **елемент *input***, у якого **атрибут *type*** дорівнює *submit* або *reset*. Наприклад:

<input type="submit" value="Відправити" />

<input type="reset" value="Очистити" />

Елемент ***input*** з атрибутом ***type="image"*** дозволяє використовувати зображення як кнопку:

<input type="image" src="search.png" name="submit"/>

що на формі виглядатиме так:

Рис. 2.16. Кнопка на формі у вигляді зображення

Крім наявності зображення, в іншому ця кнопка буде аналогічна стандартній кнопці відправки даних з форми *input type="submit"* або *button type="submit"*.

2.3.4. Текстові поля

Однорядкове текстове поле створюється за допомогою **елемента *input***, в якого атрибут ***type*** має **значення *text***:

<input type="text" name="login" />

За допомогою додаткових атрибутів можна налаштувати **текстове поле:**

- ***dirname*** задає напрямок тексту;
- ***disabled*** роблять текстове поле недоступним (візуально текстове поле затіняється);
- ***maxlength***: максимально допустима кількість символів у текстовому полі;
- ***pattern***: шаблон, якому повинен відповідати текст, що вводиться;
- ***placeholder*** виводить у текстове поле текст-підказку для вводу;
- ***readonly*** робить текстове поле доступним тільки для читання;
- ***required*** показує, що текстове поле обов'язково повинно мати значення;
- ***size*** встановлює ширину текстового поля у видимих символах;
- ***value***: значення, що відображатиметься, у текстовому полі, за замовчуванням.

Серед атрибутів текстового поля також відзначимо **атрибут *list***, що містить посилання на елемент *datalist*, який визначає набір значень, що з'являються у вигляді підказок при введенні курсора в текстове поле.

Атрибут *list* текстового поля вказує на ***id* елемента datalist**. Сам елемент *datalist* за допомогою вкладених елементів *option* визначає елементи списку, і при введенні курсора у текстове поле цей список відображається у вигляді підказки.

Наприклад:

```
<form>
 <p>
   <input type="text" name="userName"
      placeholder="Введіть ім`я"  size="18"/>
   <input type="text" name="userPhone"
      placeholder="Номер телефону"
      size="18" maxlength="11" />
   <input list="phonesList" type="text" name="model"
      placeholder="Введіть модель" />
 </p>
 <p>
   <input type="password"  name="password"
```

```
            value="Abcd-123"  size="18"/>
        <input type="search" name="term"
               value="поле пошуку" />
    </p>
    <p>
        <button type="submit">Відправити</button>
        <button type="reset">Очистити</button>
    </p>
</form>
<datalist id="phonesList">
        <option value="Xiaomi" label="Redmi 8"/>
        <option value="Huawei">35000</option>
        <option value="Honor 8A"/>
</datalist>
```

Для текстового поля вводу телефону встановлюються атрибути *maxlength* і *size*, при цьому *size* – кількість символів, які поміщаються у видимий простір поля більше, ніж допустима кількість символів *(maxlength)*. Однак ввести більше символів, ніж *maxlength*, ми не зможемо.

Отже, вищеперераховані елементи на формі мають вигляд:

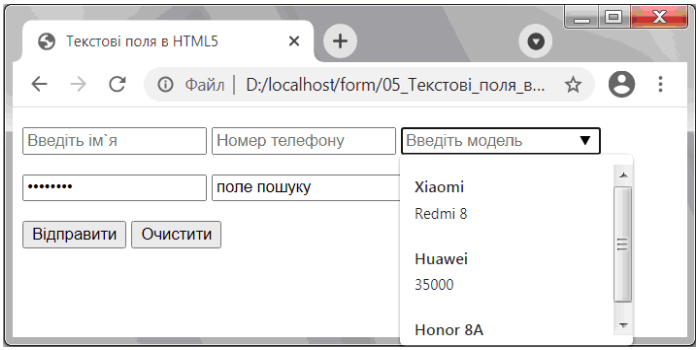


Рис. 2.17. Текстові поля на формі

2.3.5. Мітки та автофокус

Разом з полями вводу часто використовуються **текстові мітки**, які представлені **елементом *label*** (англ. мітка). Мітки створюють анотацію або заголовок до поля вводу, вказують, для чого це поле призначено. За своїм виглядом мітка нічим не відріз-

няється від звичайного тексту, але завдяки їй користувач може вибрати елемент форми кліком по тексту, який розташований в межах елемента *label*, а не по самому елементу *input*.

Для зв'язку з полем вводу мітка має **атрибут *for***, який вказує на *id* поля вводу. Наприклад,

```
<form>
  <p><label for="login">Логін: </label>
       <input type="text" id="login" name="login" />
  </p>
  <p><label for="password">Пароль: </label>
       <input type="password"
              id="password" name="password"/>
  </p>
  <p><button type="submit">Відправити </button></p>
</form>
```

У прикладі, текстове поле має атрибут *id="login"*, тому у пов'язаної з ним мітки встановлено атрибут *for="login"*. Натискання на цю мітку дозволяє перевести фокус на текстове поле для вводу логіну:

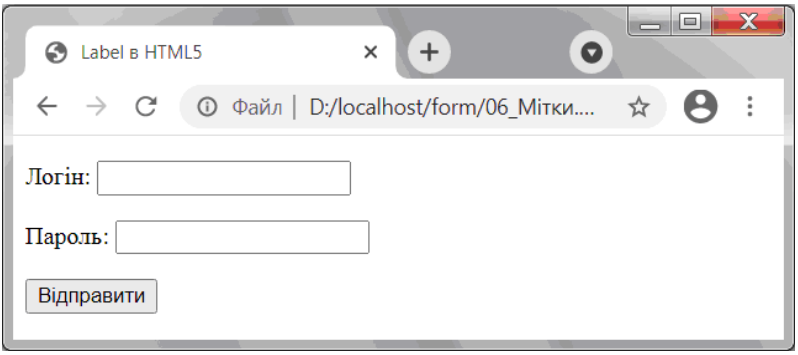


Рис. 2.18. Мітки та поля вводу на формі

Також можна встановити **автофокус**, за замовчуванням, на будь-яке поле вводу. Для цього використовується **атрибут *autofocus***:

```
<input type="text" autofocus id="login" name="login"/>
```

При запуску сторінки фокус одразу переходитиме на текстове поле.

2.3.6. Елементи для вводу чисел

Для вводу чисел використовується

1. Елемент *input* з атрибутом *type="number"*, що створює **просте числове поле**, яке можна налаштувати за допомогою наступних атрибутів:

- ***min***: мінімально допустиме значення;
- ***max***: максимально допустиме значення;
- ***readonly***: поле доступно тільки для читання;
- ***required***: вказує, що дане поле обов'язково повинно мати значення;
- ***step***: число, на яке буде збільшуватися значення в полі;
- ***value***: значення за замовчуванням.

2. Як і у випадку з текстовим полем, за допомогою атрибуту *list* можна прикріпити список *datalist* **для вибору з переліку можливих значень**.

3. **Повзунок** являє собою шкалу, на якій можна вибрати одне зі значень. Для створення повзунка використовується елемент *input* з атрибутом *type="range"*. Багато в чому повзунок схожий на просте поле для вводу чисел, також має атрибути *min*, *max*, *step* і *value*.

Приклад. Використаємо вищеперераховані елементи:

```
<form>
  <p><label for="age">Числове поле </label>
       <input type="number" step="1" min="1" max="100"
            value="10" id="age" name="age"/>
  </p>
  <p><label for="price">Вибір з переліку </label>
       <input type="number" list="priceList"
            step="1000" min="3000" max="100000"
            value="10000" id="price" name="price"/>
  </p>
  <p><label for="price">Повзунок</label>  1
       <input type="range" step="1" min="0" max="100"
            value="10" id="price" name="price"/>100
  </p>
</form>
```

Створена форма з полями для вводу чисел має вигляд:

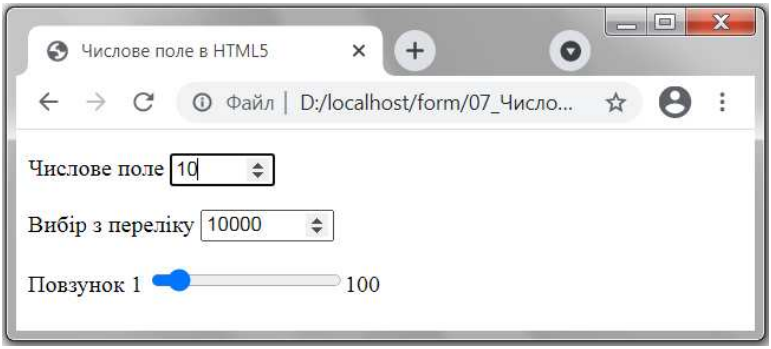


Рис. 2.19. Елементи для вводу/вибору чисел

Числове поле, за замовчуванням, має значення 10 (*value="10"*), мінімально допустиме значення, яке можна ввести: 1, а максимальне допустиме значення – 100. Атрибут *step="1"* означає, що значення буде збільшуватися на одиницю.

Залежно від браузера візуалізація цих полів може відрізнятися.

2.3.7. Прапорці та перемикачі

1. Прапорець являє собою елемент, який може знаходитися у двох станах: відміченому і невідміченому. Прапорець створюється за допомогою **елемента *input*** з атрибутом ***type="checkbox"***. Атрибут *checked* дозволяє встановити прапорець у відмічений стан.

2. Перемикачі або радіокнопки, схожі на прапорці, також можуть перебувати у відміченому або невідміченому стані, але з перемикачів можна створити групу, у якій одночасно можна вибрати тільки один перемикач.

Для створення радіокнопки потрібно для елемента *input* вказати **атрибут *type="radio"***, і тоді **атрибут *name*** означає вже не ім'я елемента, а ім'я групи, до якої належить елемент-радіокнопка.

Щоб відмітити радіокнопку, для неї встановлюється **атрибут *checked***. Важливе значення відіграє **атрибут *value***, який при відправці форми дозволяє серверу визначити, який саме перемикач був відміченим.

Наприклад, розмістимо на формі: прапорці, щоб забезпечити можливість множинного вибору, та групу радіокнопок, де можна вибрати тільки один перемикач:

```
<form>
  <h2> Позначте технології, які ви вивчаєте </h2>
  <p><input type="checkbox" checked name="html5"/>
        HTML5</p>
  <p><input type="checkbox" name="dotnet"/>
        .NET</p>
  <p><input type="checkbox" name="java"/>
        Java</p>
  <h2>Виберіть технологію</h2>
  <p><input type="radio" value="html5" checked
        name="tech"/>HTML5</p>
  <p><input type="radio" value="net" name="tech"/>
        .NET</p>
  <p><input type="radio" value="java" name="tech"/>
        Java</p>
</form>
```

У цьому випадку отримуємо

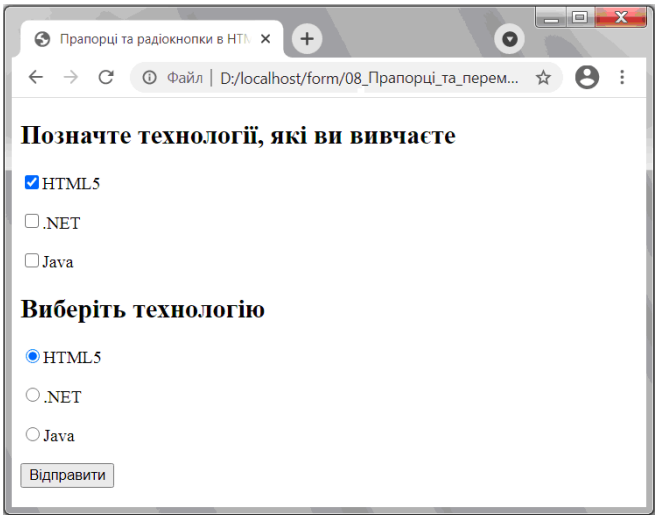


Рис. 2.20. Прапорці та перемикачі на формі

2.3.8. Елементи для вводу кольору, url, email, телефону

1. Вибір кольору

За вибір кольору в HTML5 відповідає спеціальний **елемент *input* типу *color*:**

```
<form>
   <label for="favcolor1">Виберіть колір</label>
   <input type="color" id="favcolor1" name="favcolor1" />
</form>
```

Елемент відображає обраний колір, а при натисканні на нього з'являється спеціальне діалогове вікно для вибору кольору. Значенням цього елемента буде числовий шістнадцятковий код вибраного кольору.

За допомогою **елемента *datalist*** можна задати набір колорів, з яких користувач може вибрати потрібний:

```
<form>
<label for="favcolor2">Виберіть колір</label>
<input type="color" list="colors"
       id="favcolor2" name="favcolor2"/>
<datalist id="colors">
   <option value="#0000FF" label="blue">
   <option value="#008000" label="green">
   <option value="#ff0000" label="red">
</datalist>
</form>
```

На формі ці елементи мають вигляд:

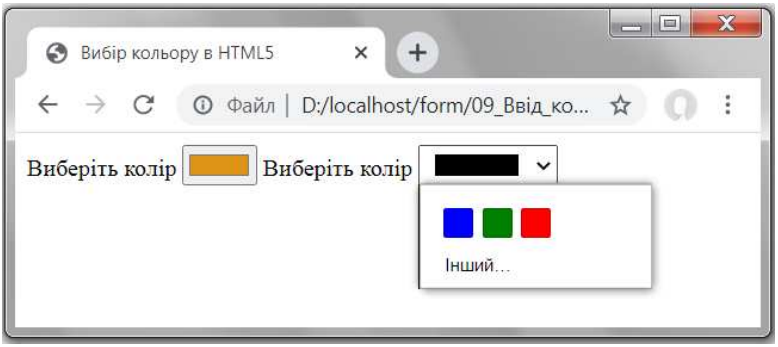


Рис. 2.21. Вибір кольору в HTML

Кожен елемент *option* в *datalist* приймає значення – шістнадцятковий код кольору, наприклад, "*#0000FF*". Після вибору кольору цей числовий код встановлюється значенням атрибуту *value* в елементі *input*.

2. Поля для вводу url, email, телефону

Ряд полів *input* призначені для вводу таких даних, як url, адреса електронної пошти та номер телефону. Вони однотипні і більшою мірою відрізняються лише тим, що **атрибут *type*** приймає відповідно значення ***email, tel* і *url***.

Для їх налаштування можна використовувати ті ж атрибути, що й для звичайного текстового поля:

- ***maxlength***: максимально допустима кількість символів у полі;
- ***pattern***: шаблон, якому повинен відповідати текст, що вводиться;
- ***placeholder***: текст, який відображатиметься в полі як підказка вводу;
- ***readonly***: робить текстове поле доступним тільки для читання;
- ***required***: вказує, що текстове поле обов'язково повинно мати значення;
- ***size***: встановлює ширину поля у видимих символах;
- ***value***: встановлює значення для поля за замовчуванням;
- ***list***: встановлює прив'язку до елемента *datalist* зі списком можливих значень.

Наприклад, визначимо на формі поля для вводу email, телефону та url:

```
<form>
  <p><label for="email">Email: </label>
    <input type="email" placeholder="user@gmail.com"
          id="email" name="email"/>
  </p>
  <p><label for="url">URL: </label>
    <input type="url" id="url" name="url"/>
  </p>
  <p><label for="phone">Телефон: </label>
    <input type="tel" placeholder="(XXX)-XXX-XXXX"
```

```
        id="phone" name="phone"/>
  </p>
  <p><button type="submit">Відправити</button></p>
</form>
```

Створена форма має вигляд:

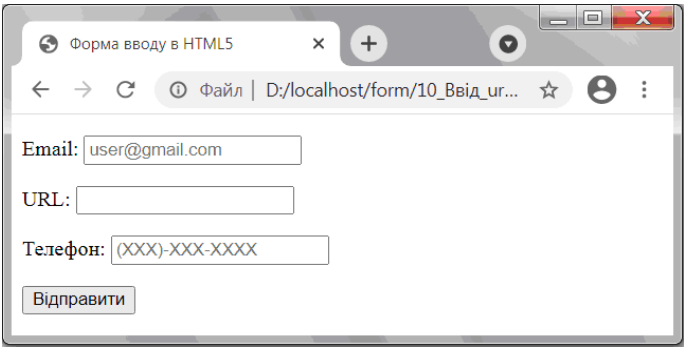


Рис. 2.22. Поля для вводу url, email, телефону

Основна перевага таких полів вводу перед звичайними текстовими полями полягає в тому, що вони використовують шаблон для перевірки вводу. Наприклад, якщо ми введемо в якесь поле некоректне значення і спробуємо відправити форму, то браузер відобразить повідомлення про некоректно введені дані і форма не буде відправлена.

2.3.9. Елементи для вводу дати і часу

Для роботи з датою і часом в HTML5 призначено **кілька типів елемента *input***, що визначаються наступними значеннями для атрибуту *type*:

- ***datetime-local***: встановлює дату і час;
- ***date***: встановлює дату;
- ***month***: встановлює поточний місяць і рік;
- ***time***: встановлює час;
- ***week***: встановлює поточний тиждень.

Використаємо вищеперелічені поля:

```
<form>
  <label for="date">Дата </label>
```

```
  <input type="date" id="date" name="date"/>
  <p><label for="week">Тиждень: </label>
      <input type="week" name="week" id="week" />
  </p>
  <p><label for="localdate">Дата і час: </label>
     <input type="datetime-local"
            id="localdate" name="date"/>
  </p>
  <p><label for="month">Місяць: </label>
      <input type="month" id="month" name="month"/>
  </p>
  <p><label for="time">Час: </label>
      <input type="time" id="time" name="time"/>
  </p>
  <p><button type="submit">Відправити</button></p>
</form>
```

Створена форма з перерахованими полями має вигляд:

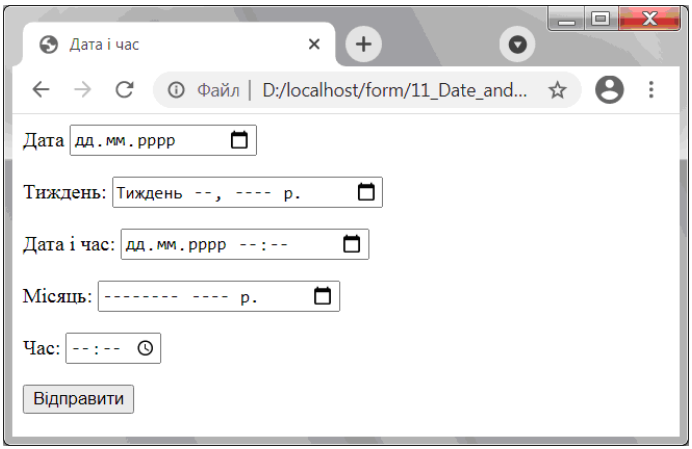


Рис. 2.23. Елементи для вводу дати і часу

При вводі в ці поля відкриватимуться відповідні календарі.

Використання цих елементів залежить від браузера, зокрема, Firefox підтримує тільки елементи *date* і *time*, для інших створюються звичайні текстові поля, а IE11 і зовсім не підтримує ці елементи.

2.3.10. Відправка файлів

За вибір файлів на формі відповідає **елемент *input*** з атрибутом ***type="file"***. Для відправки файлу на сервер форма повинна мати атрибут *enctype="multipart/form-data"*.

Елемент вибору файлу можна додатково налаштувати за допомогою набору атрибутів:

- ***accept***: встановлює тип файлів, які допустимі для вибору;
- ***multiple***: дозволяє вибрати декілька файлів одночасно;
- ***required***: вимагає обов'язкового вибору файлу.

Приклад. Розмістимо на формі елементи для одиничного та множинного вибору файлів:

```
<form enctype="multipart/form-data" method="post"
        action="http://localhost:8080/postfile.php">
  <p><label for="file1">Одиничний вибір файлу: </label>
      <input type="file" name="file1" id="file1"/>
  </p>
  <p><label for="file2">Множинний вибір файлів: </label>
      <input type="file" name="file2" id="file2" multiple />
  </p>
  <p><input type="submit" value="Відправити"/></p>
</form>
```

Елементи для вибору файлу/ів на формі мають вигляд:

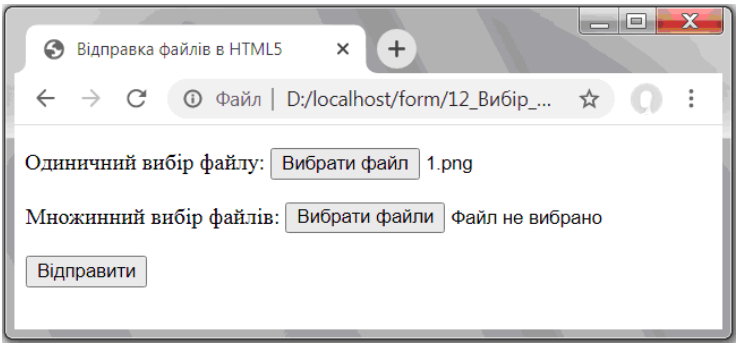


Рис. 2.24. Елементи для вибору файлів

При натисканні на кнопку "*Вибрати файл*" відкривається діалогове вікно для вибору файлу, і після вибору поруч із кнопкою відображається ім'я обраного файлу. У випадку множинно-

го вибору, щоб вибрати кілька файлів, потрібно утримувати клавішу *Ctrl* або *Shift*, а після вибору поруч із кнопкою відобразиться кількість вибраних файлів.

2.3.11. Список select

Елемент *select* створює список. Залежно від налаштувань, це може бути випадаючий список для вибору одного елемента або розгорнутий список, в якому можна вибрати одночасно кілька елементів.

У елементі *select* розміщують елементи *option* – елементи списку. Кожен **елемент *option*** містить атрибут *value*, який зберігає значення елемента списку, яке буде відправлено на сервер у разі вибору цього елемента. При цьому значення елемента *option* (атрибут *value*) не обов'язково повинно збігатися з текстом, що відображається як елемент списку.

Елемент *option* має необов'язкові (додаткові) атрибути:

1. Атрибут *selected* задає обраний за замовчуванням елемент, це не обов'язково перший елемент у списку.

2. Атрибут *disabled* забороняє вибір певного елемента; як правило, елементи із цим атрибутом служать для створення заголовків.

Приклад. Створимо випадаючий список

```
<form method="get">
  <p>
    <select id="phone" name="phone">
      <option disabled selected>Виберіть модель
        </option>
      <option value="SamsA50">Samsung Galaxy A50
        </option>
      <option value="Xiaomi10">Xiaomi Mi Note 10
        </option>
      <option value="HuaweiPSmart">Huawei P Smart
        </option>
    </select>
  </p>
</form>
```

Випадаючий список на формі має вигляд:

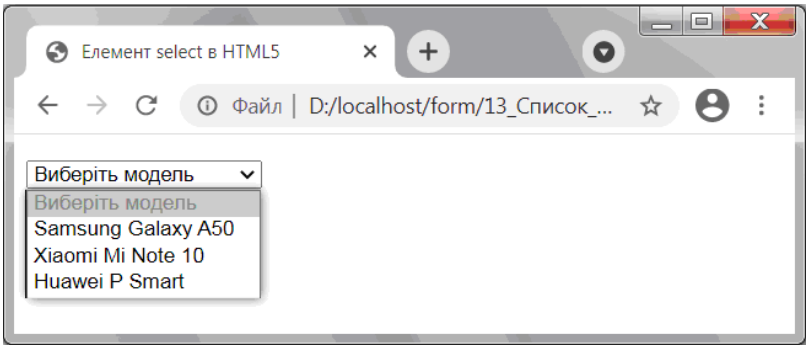


Рис. 2.25. Випадаючий список

Щоб створити **список із множинним вибором,** у визначення елемента *select* потрібно додати **атрибут *multiple***, наприклад:

*<select **multiple** id="phone" name="phone">…</select>*

Утримуючи клавішу *Shift*, можна вибрати в такому списку одночасно кілька елементів.

Елемент *select* також дозволяє **групувати елементи** за допомогою **тега *optgroup***:

```
<form method="get">
  <select id="phone" name="phone">
    <option disabled selected>Виберіть модель</option>
      <optgroup label="Samsung">
        <option value="A50">GalaxyA50
          </option>
        <option value="Note10Plus">Galaxy Note 10 Plus
          </option>
        <option value="S10+">Galaxy S 10+
          </option>
      </optgroup>
      <optgroup label="Xiaomi">
        <option value="MiNote10">Mi Note 10</option>
        <option value="Redmi8">Redmi 8 </option>
      </optgroup>
  </select>
</form>
```

Створений список із групуванням елементів має вигляд:

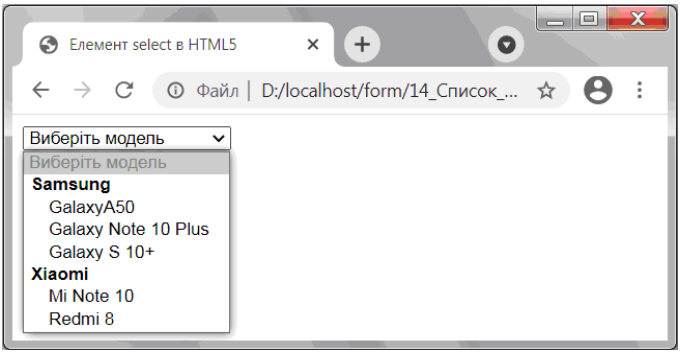


Рис. 2.26. Список з групуванням елементів

Використання груп елементів може бути застосовано як до випадаючого списку, так і до списку з множинним вибором.

2.3.12. Багаторядкове текстове поле textarea

Елемент ***<input type="text"/>*** дозволяє створити просте **однорядкове текстове поле**.

Якщо можливостей цього елемента для вводу тексту недостатньо, то можна використовувати багаторядкове текстове поле, представлене **елементом *textarea***.

За допомогою додаткових **атрибутів *cols*** і ***rows*** можна задати, відповідно, кількість стовпців і рядків:

```
<form method="get">
  <p>
    <label for="comment">Ваш коментар:</label>
    <br/>
    <textarea id="comment" name="comment"
        placeholder="Написати коментар"
        cols="30"
        rows="7">
    </textarea>
  </p>
</form>
```

Створена форма з багаторядковим текстовим полем:

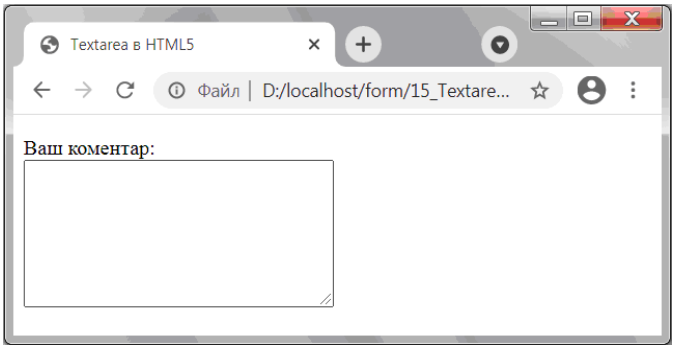


Рис. 2.27. Багаторядкове текстове поле *Textarea*

2.3.13. Валідація форм

В нашому розпорядженні є різні елементи, які можна використовувати на формі для вводу різноманітних значень та даних. Однак часто користувачі вводять не зовсім коректні значення, наприклад, очікується ввід чисел, а користувач вводить букви і т.д. Для попередження та перевірки некоректного вводу **в HTML5 існує механізм валідації**.

Перевага використання валідації в HTML5 полягає в тому, що користувач після некоректного вводу може одразу отримати повідомлення про помилку і внести відповідні зміни у введені дані.

Для створення валідації елементів форм HTML5 використовуються атрибути:

• ***required*** вимагає обов'язкового вводу значення для елементів *textarea*, *select*, *input* (із типом *text*, *password*, *checkbox*, *radio*, *file*, *datetime-local*, *date*, *month*, *time*, *week*, *number*, *email*, *url*, *search*, *tel*);

• ***min*** і ***max:*** мінімально та максимально допустимі значення для елемента *input* з типом *datetime-local*, *date*, *month*, *time*, *week*, *number*, *range*;

• ***pattern*** задає шаблон, якому повинні відповідати дані, що вводяться; для елемента *input* з типом *text*, *password*, *email*, *url*, *search*, *tel*.

Приклад. Розмістимо на формі поля для логіну та паролю з вимогою обов'язкового вводу; поле для вводу віку з обмежен-

нями діапазону значень; поле для вводу телефону згідно з шаблоном:

```
<form method="get">
  <p><label for="login">Логін:</label>
      <input type="text" required id="login" name="login"/>
  </p>
  <p> <label for="password">Пароль:</label>
       <input type="password" required
           id="password" name="password"/>
  </p>
  <p><label for="age">Вік:</label>
     <input type="number" min="1" max="100" value="18"
           id="age" name="age"/> 
     <label for="phone">Телефон:</label>
     <input type="text" placeholder="+1-234-567-8901"
      pattern="\+\d-\d{3}-\d{3}-\d{4}" id="phone"
      name="phone"/>
  </p>
  <p><input type="submit" value="Відправити" /></p>
</form>
```

Створена форма має такий вигляд:

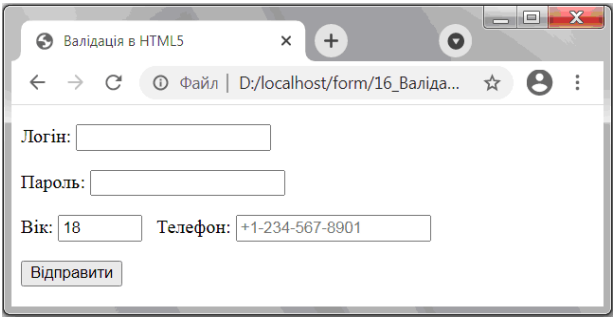


Рис. 2.28. Валідація полів форми

Атрибут *required* вимагає обов'язкової наявності значення. Якщо залишити поля з атрибутом *required* порожніми, то при натисканні на кнопку відправки даних, браузер виведе на екран повідомлення з вимогою заповнення поля і дані не будуть відправлені на сервер.

Для обмеження діапазону значень, що вводяться, використовуються **атрибути *min*** і ***max***.

Атрибут *pattern* задає шаблон, якому повинні відповідати дані. Для визначення шаблону використовується мова регулярних виразів. Розглянемо регулярний вираз, що використовується для вводу номера телефону:

\+\d-\d{3}-\d{3}-\d{4}.

Вираз \+ означає, що першим елементом у номері повинен бути знак плюс “+”.

Вираз \d означає будь-яку цифру від 0 до 9.

Вираз \d{3} означає три цифри підряд, а \d{4} – чотири цифри підряд, тобто цей вираз буде відповідати номеру телефону в форматі "+1-234-567-8901".

Якщо ввести дані, які не відповідають цьому шаблону, і натиснути на кнопку відправки даних, то браузер відобразить повідомлення про помилку.

Відключення валідації

Не завжди валідація є бажаною, іноді потрібно її відключити. В цьому випадку можна використати або **атрибут *novalidate* елемента форми**, або **атрибут *formnovalidate* кнопки відправки**:

```
<form novalidate method="get">
…
  <input type="submit" value="Відправити"
         formnovalidate />
…
</form>
```

2.3.14. Елементи fieldset і legend

Для групування елементів форми часто використовується **елемент *fieldset***, який створює межі навколо вкладених елементів, візуально створюючи з них групу. Разом з ним використовується **елемент *legend***, який задає заголовок для групи елементів.

При необхідності можна створити на одній формі кілька груп за допомогою елементів *fieldset*.

Приклад. Згрупуємо елементи форми для проходження авторизації:

```
<form>
    <fieldset>
      <legend>Введіть дані</legend>
      <label for="login">Логін:</label><br>
      <input type="text" name="login" id="login"/><br>
      <label for="password">Пароль:</label><br>
      <input type="password"
             name="password" id="password"/><br>
      <input type="submit" value="Авторизація">
    </fieldset>
</form>
```

Створена форма має вигляд:

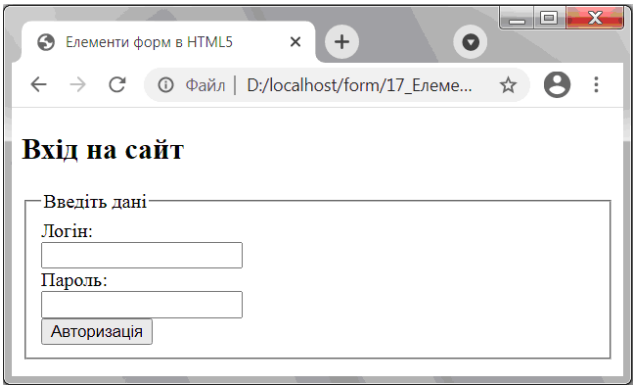


Рис. 2.29. Елементи *fieldset* і *legend*

2.4. Семантична структура сторінки

Усі HTML – теги, які ми досі розглядали, використовуються, переважно, для форматування контенту: вони повідомляють браузеру, як показувати контент на сторінці, але не дають визначення типу вмістимого (контенту) і не вказують, яку роль відіграє контент на сторінці.

Семантичний HTML5 усуває цей недолік, визначаючи теги для пояснення чіткої ролі контенту на сторінці. Ця додаткова інформація допомагає роботам-індексаторам, таким як Google і Bing, краще зрозуміти, який контент важливий, який другорядний, який використовується для навігації і так далі. Використо-

вуючи семантичні HTML-теги, розробник надає додаткову інформацію, яка допомагає пошуковим системам розуміти ролі і відносну важливість різних частин веб-сторінок.

Елементи, що використовуються для створення семантичної структури веб-сторінки:

1. Елемент *article* представляє цілісний блок інформації на сторінці, який може розглядатися окремо і використовуватися незалежно від інших блоків. Наприклад, це може бути пост на форумі або стаття в блозі, онлайн-журналі, коментар користувача.

Один елемент *article* може містити кілька вкладених елементів *article*. Наприклад, можна оформити в елемент *article* всю статтю в блозі, і цей елемент буде містити вкладені елементи *article*, які представляють коментарі до цієї статті в блозі. Тобто стаття в блозі може розглядатися нами як окрема семантична одиниця, і одночасно коментарі також можна розглядати окремо, незалежно від будь-якого вмісту.

При використанні *article* треба враховувати, що кожен елемент *article* повинен бути ідентифікований за допомогою включення в нього одного із заголовків *h1-h6*.

Наприклад, веб-сторінка містить елемент *article*, в якому розміщені стаття (текстове наповнення) та набір коментарів, кожен з яких офомлений в елемент *article*.

2. Елемент *section* об'єднує пов'язані між собою частини *html*-документа, виконуючи їх групування. Наприклад, *section* може включати набір вкладок на сторінці, новини, об'єднані за категоріями, і т.д.

Кожен елемент *section* повинен бути ідентифікований за допомогою одного із заголовків *h1-h6*.

При цьому елемент *section* може містити кілька елементів *article*, виконуючи їх групування, аналогічно як елемент *article*, може містити кілька елементів *section*.

Приклад. Веб-сторінка має таку будову: для блоку основного вмісту створюється секція і для набору коментарів окремо створюється елемент *section*, в якому кожен коментар оформляється в елемент *article*.

3. Елемент *nav* містить набір посилань, як правило, у вигляді ненумерованого списку, що є навігацією по сайту. Такі по-

силання зазвичай ведуть на інший розділ, головну сторінку, контакти та інше.

На одній веб-сторінці можна використовувати кілька елементів *nav*. Наприклад, один елемент навігації для переходу по сторінках на сайті, а інший – для переходу всередині *html*-документа.

Не всі посилання обов'язково розміщати в елемент *nav*. Тобто деякі посилання можуть не являти собою зв'язний блок навігації, наприклад, посилання на головну сторінку, на ліцензійну угоду з приводу використання сервісу і подібні посилання, які часто розміщуються внизу сторінки. Як правило, їх досить визначити в елементі *footer*, а елемент *nav* для них використовувати необов'язково.

4. Елемент *header* являє собою вступний елемент, що передує основному вмісту, де можуть бути заголовки, елементи навігації або будь-які інші допоміжні елементи, зокрема логотип, форма пошуку і т.п.

У HTML-документі може міститися одночасно кілька елементів *header* і вони можуть розташовуватися у будь-якій частині сторінки, але елемент *header* не може розміщуватися в таких елементах, як *address*, *footer* або інший *header*.

5. Елемент *footer* задає нижній колонтитул документу або розділу; зазвичай містить інформацію про те, хто автор контенту на веб-сторінці, копірайт, дату публікації, блок посилань на схожі ресурси і т.д. Як правило, подібна інформація розташовується у кінці веб-сторінки або основного вмістимого, однак *footer* не має чіткої прив'язки до позиції і може використовуватися в різних місцях веб-сторінки.

Футер необов'язково повинен бути визначений для всієї сторінки, це може бути і окрема секція контенту.

Елемент *footer* не слід розміщати в таких елементах, як *address*, *header* або інший *footer*. Зауважимо, що контактна інформація всередині елемента *footer* має бути в контейнері *address*.

Елемент *address* призначений для відображення контактної інформації, яка пов'язана з найближчим елементом *article* або *body*. Часто цей елемент розміщується у футері.

6. Елемент *aside* представляє вміст, який опосередковано пов'язаний з іншим контентом на веб-сторінці і може розглядатися незалежно від нього. Цей елемент можна використовувати, наприклад, для сайдбарів, для рекламних блоків, блоків елементів навігації, різних плагінів типу твіттера або фейсбук і т.д.

7. Елемент *main* представляє основний вміст веб-сторінки, тобто її унікальний контент, в який не слід включати повторювані на різних веб-сторінках елементи сайдбарів, навігаційні посилання, інформацію про копірайт, логотипи тощо. На веб-сторінці допустима наявність тільки одного елемента *main*.

Не варто вважати, що абсолютно весь вміст потрібно обов'язково розміщувати в елемент *main*. Можна використовувати поза ним інші елементи, наприклад *header* і *footer*. Зверніть увагу, що елемент *main* не може бути вкладеним в такі елементи, як *article*, *aside*, *footer*, *header*, *nav*.

Припустимо, що потрібно зверстати сайт згідно з таким макетом:

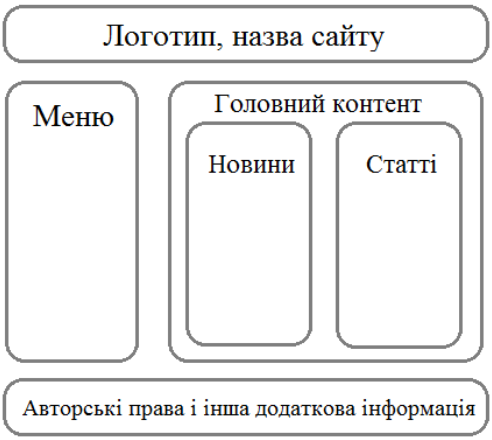


Рис. 2.30. Макет сторінки сайту

Зверставши його на HTML5, отримаємо такий макет (рис. 2.31):

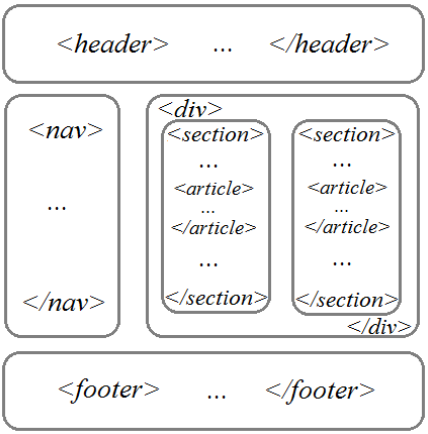


Рис. 2.31. Макет сторінки сайту на HTML5

Отже, очевидно, що верхній елемент це <*header*>. Збоку розташовується меню сайту, це елемент <*nav*>, центральний контент все ж оформлений в <*div*>. Виникає питання: чому не <*section*> або не <*article*>? Тому що це не є окремий розділ сайту, він не має заголовку – це лише місце для основного контенту.

А от вже вкладені розділи “Новини” і “Статті” цілком підходять для формування тегом <*section*>. В середині них матимемо набір новин і статтей, які можна оформити в теги <*article*>. Останній елемент на сторінці елемент <*footer*>.

Тепер сторінка має логічну структуру.

Зазначимо, що бокове меню може бути сформовано інакше, якщо бокова панель містить не одне меню, а декілька або ще якусь рекламу, банери тощо, тоді варто вкласти це вмістиме в тег <*aside*>:

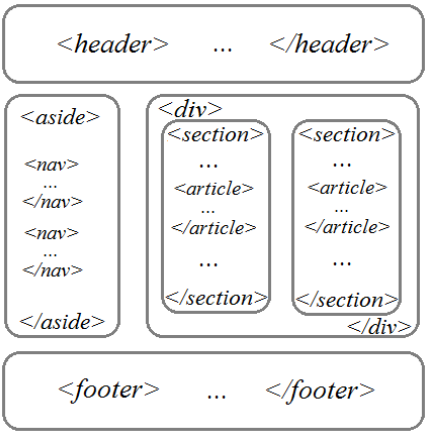


Рис. 2.32. Оформлення бокової панелі

3. ТАБЛИЦІ КАСКАДНИХ СТИЛІВ CSS3

3.1. Основи CSS3. Селектори

3.1.1. Поняття про стилі

Каскадні таблиці стилів (**C**ascading **S**tyle **S**heets, скор. CSS), або, простіше кажучи, стилі, визначають зовнішній вигляд документа, його стилізацію.

Стиль в CSS – це правило, яке вказує браузеру, як потрібно форматувати елемент. Форматування може включати задання кольору фону елемента, кольору і типу шрифту і так далі.

Визначення стилю складається з двох частин: селектор, який визначає елементи, до яких буде застосовано стиль, і блок оголошення стилю – набір команд, які визначають правила форматування. В загальному випадку, це виглядає наступним чином: *селектор { блок оголошення стилу}*.

Наприклад:

```
div {
    background-color:red;
    width: 100px;
    height: 60px;
}
```

У даному випадку селектором є *div*. Це означає, що стиль буде застосовуватися до всіх елементів *div*.

Після селектора у фігурних дужках слідує блок оголошення стилю: між відкриваючою і закриваючою фігурними дужками перераховуються команди, які вказують, як форматувати елемент. Кожна команда складається з **властивості і значення**. Так, у виразі

background-color:red;

background-color – назва властивості, а *red* – значення.

Властивість визначає конкретний стиль. Існує велика кількість властивостей CSS. Наприклад, *background-color* визначає колір фону. Після двокрапки вказується значення цієї властивості. Наприклад, вищезазначена команда визначає для властивості *background-color* значення *red*.

Після кожної команди ставиться крапка з комою, яка відділяє цю команду від інших.

Набори таких стилів часто називають **таблицями стилів** або **CSS**.

Існують різні способи визначення стилів:

1. Атрибут *style*

Перший спосіб полягає у визначенні стилів безпосередньо у елементі за допомогою атрибута *style*:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Стилі</title>
  </head>
  <body>
    <h2 style="color:blue;">Стилі</h2>
    <div style="background-color:red;
        width:100px; height:100px;">
    </div>
  </body>
</html>
```

Результат:

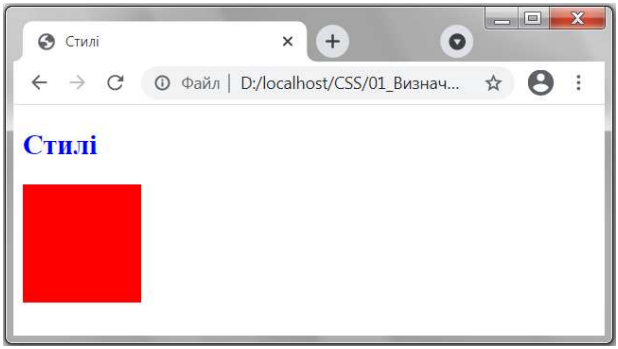


Рис. 3.1. Визначення стилів для заголовку *h2* і блоку *div*

У прикладі вище визначені два елементи – заголовок *h2* і блок *div*. Для заголовку визначено синій колір тексту за допомогою властивості *color*, для блоку *div* визначені властивості ширини (*width*), висоти (*height*), а також кольору фону (*background-color*).

2. Другий спосіб полягає **у використанні елемента *style*** в документі html, який повідомляє браузеру, що дані всередині нього є кодом CSS, а не HTML.

Часто елемент *style* визначається всередині елемента *head*, проте може також використовуватися в інших частинах HTML-документа. **Елемент *style* містить** набори стилів. Наприклад:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Стилі</title>
  <style>
   h2 {
    color:blue;
   }
   div{
    width: 100px;
    height: 100px;
    background-color: red;
   }
  </style>
 </head>
 <body>
  <h2>Стилі</h2>
  <div></div>
  </body>
</html>
```

Результат у цьому випадку буде абсолютно тим самим, що і в попередньому прикладі.

Другий спосіб робить html-код “чистішим” завдяки винесенню стилів в елемент *style*.

3. Але є і **третій спосіб**, який полягає у винесенні стилів у зовнішній файл.

Створимо в одній папці з html-сторінкою текстовий файл, який перейменуємо в ***styles.css***, і визначимо в ньому наступний вміст:

```
h2 {
   color:blue;
```

```
}
div{
   width: 100px;
   height: 100px;
   background-color: red;
}
```

Це ті ж стилі, що були всередині елемента *style*. Змінимо код html-сторінки:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Стилі</title>
  <link rel="stylesheet" type="text/css" href="styles.css"/>
 </head>
 <body>
   <h2>Стилі</h2>
   <div></div>
 </body>
</html>
```

У цьому прикладі немає елемента *style*, але є **елемент *link***, який підключає виществорений файл *styles.css*:

```
<link rel="stylesheet" type="text/css" href="styles.css"/>
```

Таким чином, визначаючи стилі у зовнішньому файлі, ми робимо код html "чистішим", структура сторінки відділяється від її стилізації. При такому визначенні, стилі набагато легше модифікувати, ніж якби вони були визначені всередині елементів або в елементі *style*. Тому цей спосіб найкращий у HTML5.

Використання стилів у зовнішніх файлах дозволяє зменшити навантаження на веб-сервер за допомогою механізму кешування. Оскільки веб-браузер може кешувати css-файл і при наступному зверненні до веб-сторінки витягувати потрібний css-файл з кешу.

4. Також можлива ситуація, коли **всі ці підходи поєднуються**: для одного елемента деякі властивості CSS визначені всередині самого елемента, інші властивості – всередині елемента *style*, а треті перебувають у зовнішньому підключеному файлі. Наприклад:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="styles.css"/>
    <style>
      div{
          width:200px;
         }
    </style>
  </head>
  <body>
     <div style="width:120px;"></div>
  </body>
</html>
```

А в файлі ***style.css*** визначений наступний стиль:

```
div {
   width:50px;
   height:50px;
   background-color:red;
}
```

У даному випадку в трьох місцях для елемента *div* визначено властивість *width*, причому з різним значенням. Яке значення буде використовуватися до елемента в результаті? В таких випадках діє наступна **система пріоритетів**:

1. Вбудовані стилі (*inline*-стилі) мають найвищий пріоритет, тобто в прикладі вище підсумковою шириною буде 120 пікселів.

2. Далі в порядку спадання пріоритету стилі, які визначені в елементі *style*.

3. Найменш пріоритетні стилі ті, які визначені в зовнішньому файлі.

3.1.2. Селектори

Раніше ми розглядали селектори тегів. При визначенні такого селектора його стиль буде застосовуватися до всіх елементів, що відповідають цьому селектору.

1. Селектор класу

У випадках, коли для одних і тих самих елементів потрібна

різна стилізація, використовуються класи. Для визначення **селектора класу** в CSS перед назвою класу ставиться крапка:

```
<style>
   .redBlock {
      background-color:red;
   }
   …
</style>
```

Назва класу може бути довільною, наприклад, у даному випадку "*redBlock*". В імені класу дозволяється використовувати літери, цифри, дефіси та знаки підкреслення. Причому назва класу обов'язково має починатися з літери. Регістр літер у назві класу має значення.

Після визначення класу можна його застосувати до елементу за допомогою атрибуту *class*. Наприклад:

```
<body>
   …
<div class="redBlock">Вмістиме елемента div</div>
   …
</body>
```

До одного і того ж елементу можна **одночасно застосувати кілька класів**. Для цього в значенні атрибуту *class* елемента перераховують назви класів, розділені пробілом:

```
<!DOCTYPE html>
<html>
 <head>
      <meta charset="utf-8">
      <title>Класи</title>
      <style>
           .redBlock { background-color:red;}
           .sizeBlock { width:100px; height:100px;}
      </style>
 </head>
 <body>
    <div class="redBlock sizeBlock"></div>
 </body>
</html>
```

При цьому значення атрибуту *class* обов'язково зазначається **в лапках**. Інакше, значенням атрибуту *class* буде тільки перший зазначений клас, а наступний – браузер буде інтерпретувати як окремий атрибут.

2. Ідентифікатори елементів

Для ідентифікації унікальних на веб-сторінці елементів використовуються **ідентифікатори елементів**, які визначаються за допомогою **атрибуту *id***. Наприклад, на сторінці головний блок або шапка:

```
<div id="header">…</div>
```

Визначення стилів для ідентифікаторів елементів аналогічно визначенню класів, але замість крапки ставиться символ решітки #:

Наприклад:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ідентифікатори CSS</title>
    <style>
      div {
         margin: 10px;
         border: 1px solid #222;
        }
      #header {
         height: 80px;
         background-color: #ccc;
        }
      #content {
          height: 180px;
          background-color: #eee;
         }
      #footer {
          height: 80px;
          background-color: #ccc;
         }
    </style>
  </head>
```

```
  <body>
    <div id="header"> <h2>Шапка сайту </h2> </div>
    <div id="content"> <h2>Основне вмістиме </h2> </div>
    <div id="footer"> <h2> Футер </h2> </div>
  </body>
</html>
```

Результат:

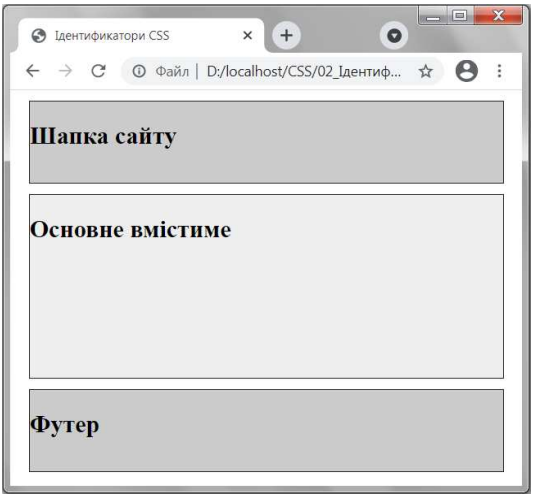


Рис. 3.2. Ідентифікатори елементів

Ідентифікатори більшою мірою відносяться до структури веб-сторінки і меншою мірою – до стилізації. Для стилізації використовуються переважно класи, а не ідентифікатори.

3. Універсальний селектор

В CSS також є так званий універсальний селектор, який представляється символом *****. Він застосовує стилі до всіх елементів на html-сторінці:

```
*{
   background-color: red;
}
```

4. Стилізація групи селекторів

Іноді певні стилі застосовуються до цілого ряду селекторів. Наприклад, відобразити всі заголовки на веб-сторінці червоним

кольором. У цьому випадку можна перерахувати селектори всіх елементів через кому:

```
h1, h2, h3, h4{
        color: red;
  }
```

Група селекторів може містити як селектори тегів, так і селектори класів та ідентифікаторів. Наприклад:

```
h1, #header, .redBlock{
   color: red;
  }
```

3.1.3. Селектори нащадків (усі рівні вкладеності)

Веб-сторінка може мати складну організацію, одні елементи можуть містити інші. Вкладені елементи також можна назвати **нащадками**, а контейнер цих елементів – **батьком (або пращуром)**.

Використовуючи спеціальні селектори, можна стилізувати вкладені елементи або нащадки всередині визначених елементів.

Для застосування стилю до вкладеного елементу всіх рівнів вкладеності селектор повинен містити спочатку батьківський елемент, а потім вкладений, розділені пробілом:

```
#main p{ font-size: 16px; }
```

Тобто цей стиль буде застосовуватися тільки до тих елементів *p*, які знаходяться всередині елемента з ідентифікатором *main*.

Наприклад, на сторінці можуть бути параграфи всередині блоку з основним вмістом і всередині блоку футера. Завдання полягає в тому, щоб для параграфів всередині блоку основного вмісту встановити один шрифт, а для параграфів футера – інший:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Селектори CSS</title>
    <style>
      #main p{
          font-size: 18px;
```

```
        }
      #footer p{
          font-size: 15px;
        }
    </style>
  </head>
  <body>
    <div id="main">
      <div><p>Перший абзац</p></div>
      <p>Другий абзац</p>
    </div>
    <div id="footer">
      <p>Текст футера</p>
    </div>
  </body>
</html>
```

Результат застосування стилів:

Рис. 3.3. Селектори нащадків (усі рівні вкладеності)

Також можна застосувати стиль до елементів, які містяться всередині іншого елемента та зазначені з атритубум ***class***. Наприклад:

```
<style>
    li .redLink{ color: red; }
</style>
```

У цьому випадку стиль буде застосований до елементів, які містяться всередині елемента *<li>* та зазначені з атритубум

class="redLink", напр. до елемента *<a>…* у виразі:

* Apple: 12
*

Зверніть увагу на **пробіл у визначенні стилю**: “*li .redLink*”, який вказує, що елементи з атрибутом *class="redLink"* повинні бути вкладеними по відношенню до елемента *<li>*. У випадку, **якщо прибрати пробіл**:

li.redLink{ color: red; }

зміст селектора зміниться і стиль буде застосовано **до елементів *<li>* з атрибутом *class="redLink"***. Наприклад, до наступного елемента:

```
<li class="redLink">
  Microsoft: <a>Lumia 650</a>
</li>
```

3.1.4. Селектори дочірніх елементів (перший рівень вкладеності)

Селектори дочірніх елементів відрізняються від селекторів нащадків тим, що дозволяють вибрати елементи тільки першого рівня вкладеності.

Для звернення до дочірніх елементів використовується **знак кутової дужки** (напр., селектор *article > p{ color: red; }* діє тільки на ті параграфи, які знаходяться безпосередньо в елементі *article* на першому рівні вкладеності).

Зауважимо: якщо використовути селектор **без символу >**:

article p{ color: red; }

то стиль буде застосований до всіх параграфів на всіх рівнях вкладеності.

Приклад використання селекторів нащадків та дочірніх елементів:

```
<!DOCTYPE HTML>
<html>
 <head>
  <meta charset="utf-8">
  <title> Селектори нащадків та дочерні селектори </title>
  <style>
   div i { /* селектор нащадка */
    color: green; /* зелений колір тексту */
    text-decoration: underline;
```

```
 }
 p > i { /* дочірній селектор */
  color: red; /* червоний колір тексту */
  text-transform: uppercase; /* верхній регістр символів */
 }
 </style>
</head>
<body>
 <div>
  <i>Lorem ipsum dolor sit amet, consectetur adipiscing elit,
  sed do eiusmod tempor incididunt ut labore et dolore magna
  aliqua</i>
  <p><i>Lorem ipsum dolor sit amet </i>, consectetuer
   adipiscing elit, sed diem nonummy nibh euismod tincidunt ut
   lacreet
   <i>dolore magna</i> aliguam erat volutpat.
  </p>
 </div>
</body>
</html>
```

Результат:

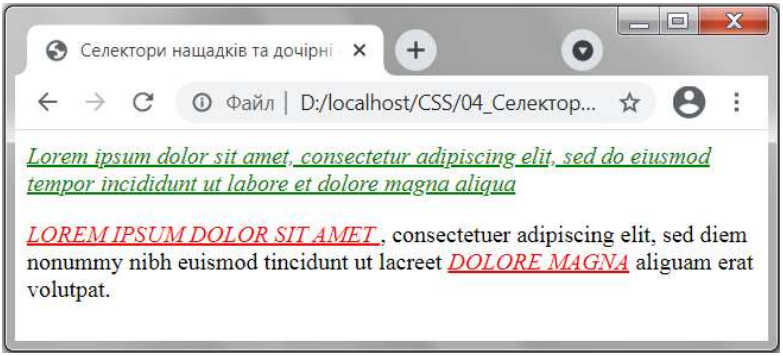


Рис. 3.4. Селектори нащадків та дочірніх елементів

3.1.5. Селектори елементів одного рівня

Селектори елементів одного рівня та **суміжних елементів** дозволяють вибрати елементи, які знаходяться на одному рівні вкладеності. Іноді такі елементи ще називають сіблінгами (siblings) або сестринськими елементами.

Суміжними (також наз. сусідніми) називаються елементи веб-сторінки, які розміщені безпосередньо один за одним в коді веб-сторінки. Розглянемо, наприклад:

* Lorem ipsum* *******dolor******* ***<var>****sit****</var>*** *amet. *

У цьому випадку теги ****** *і* ***<var>*** є суміжними елементами.

Щоб стилізувати перший суміжний елемент після певного елементу, використовується символ плюс **"+"**.

Приклад. Селектор *h2+div* дозволяє визначити стиль (у даному випадку підкреслений текст червоного кольору) для блоку *div*, який розміщений безпосередньо після заголовка *h2*:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Селектори в CSS3</title>
    <style>
      h2+div {
       color: red; text-decoration: underline;
      }
    </style>
  </head>
  <body>
    <h2>Заголовок</h2>
    <div>
      <p>Текст першого блоку (перший div після h2) </p>
    </div>
    <div>
      <p>Текст другого блоку (другий div після h2) </p>
    </div>
  </body>
</html>
```

Результат:

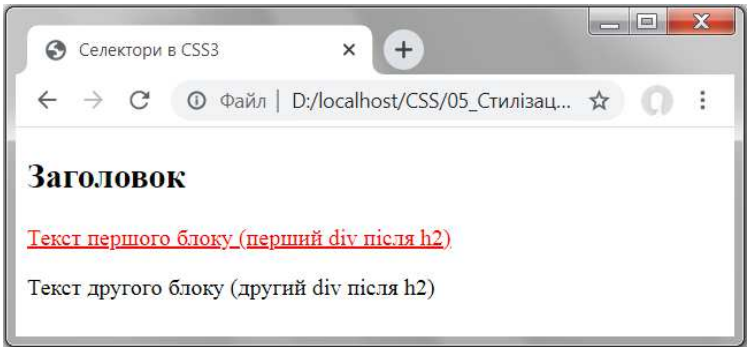


Рис. 3.5. Стилізація першого суміжного елемента

Причому селектор *h2+div* буде стилізувати блок *div*, тільки якщо він буде йти безпосередньо після заголовка *h2*; якщо ж між заголовком і блоком *div* буде перебувати будь-який інший елемент, то стиль не буде застосовуватися.

Якщо потрібно стилізувати **всі суміжні елементи одного рівня**, неважливо безпосередньо вони йдуть після певного елемента чи ні, то в цьому випадку замість знака плюс потрібно використати знак тильди "~":

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>Селекторы в CSS3</title>
   <style>
     h2+div { color: red;  text-decoration: underline;}
     h2~div { font-weight: bold;}
   </style>
 </head>
 <body>
   <h2>Заголовок</h2>
   <p>Анотація</p>
   <div>
     <p>Текст першого блоку (перший div після h2) </p>
   </div>
   <div>
     <p>Текст другого блоку (другий div після h2) </p>
```

```
    </div>
  </body>
</html>
```

У цьому випадку селектор ***h2+div*** не спрацює, оскільки між заголовком *h2* і наступним після нього *div* знаходиться елемент * Анотація *. Селектор ***h2~div*** буде застосований до двох блоків *div*, які знаходяться на одному рівні вкладеності із заголовком *h2*.

Тому отримуємо

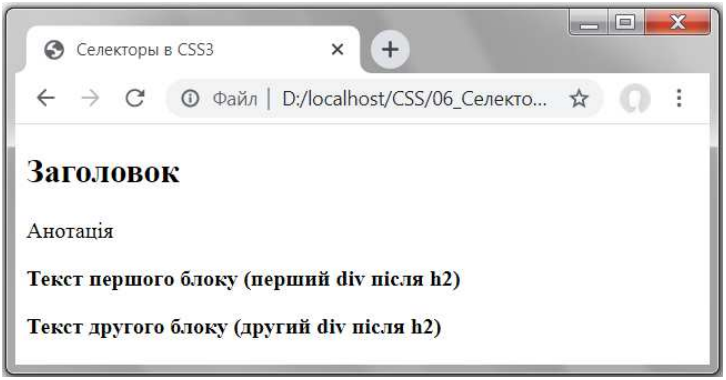


Рис. 3.6. Стилізація суміжних елементів одного рівня

3.1.6. Псевдокласи

Крім селекторів тегів, класів та ідентифікаторів, також доступний **селектор псевдокласів.** Псевдокласи надають додаткові можливості для вибору потрібних елементів. При використанні псевдокласу перед ним завжди ставитися **двокрапка**.

Список доступних псевдокласів:

:root дозволяє вибрати кореневий елемент веб-сторінки, можливо, найменш корисний селектор, оскільки на правильній веб-сторінці кореневим елементом фактично завжди є елемент *<html>*;

:link застосовується до посилання у звичайному стані, по якому ще не здійснено перехід (***до переходу)***;

:visited застосовується до посилання, по якому користувач вже виконав перехід (***після переходу)***;

:active застосовується до посилання в момент, коли користувач здійснює по ньому перехід (***в момент переходу)***;

:hover застосовується до елемента, на який користувач навів покажчик миші; використовується в основному до посилань, однак може бути застосований й до інших елементів, напр. параграфів;

:focus застосовується до елемента, який отримує фокус, коли користувач натискає кнопку табуляції або натискає кнопкою миші на полі вводу (наприклад, текстове поле);

:not дозволяє виключити елементи зі списку елементів, до яких застосовується стиль;

:lang стилізує елементи на підставі значення атрибуту *lang*;

:empty вибирає порожні елементи (напр. *<input type="text"/>*), а також елементи, які не містять вкладених елементів, тексту або пробілів (напр.: *<form></form>*, * *).

Наприклад, стилізуємо посилання, використовуючи псевдокласи:

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>Псевдокласи в CSS3</title>
   <style>
    a:link    {color:blue; text-decoration:none}
    a:visited {color:pink; text-decoration:none}
    a:hover {color:red; text-decoration:underline}
    a:active  {color:yellow; text-decoration:underline}
    input:hover {border:2px solid red;}
    a:not(.blueLink) {color: red; font-weight: bold;}
   </style>
 </head>
 <body>
   <a href="index.html" class="blueLink">
      Навчання CSS3</a>
   <input type="text"/>
   <br><br><br>
   <a>Перше посилання</a><br/>
```

```
    <a class="blueLink">Друге посилання</a><br/>
    <a>Третє посилання</a>
  </body>
</html>
```

Результат:

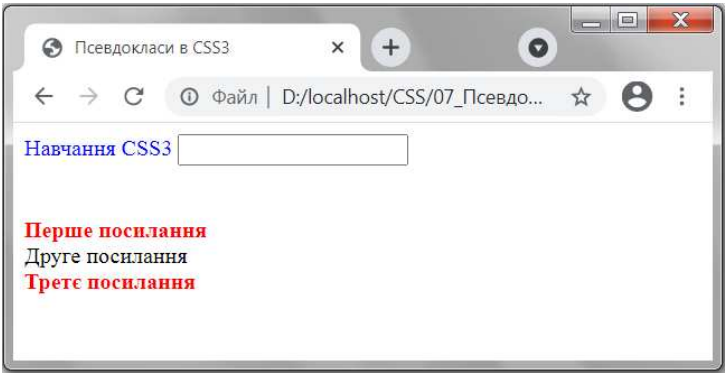


Рис. 3.7. Використання псевдокласів для стилізації посилань

Селектор *a:not(.blueLink)* застосовує стиль до всіх посилань, крім тих, які з атрибутом *class="blueLink"*, в дужках вказується селектор елементів, які потрібно виключити.

Селектор *:lang* вибирає елементи на підставі атрибуту *lang*:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Селектори в CSS3</title>
    <style>
      :lang(uk) { color: red; text-decoration: underline;}
    </style>
  </head>
  <body>
   <form>
     <p lang="uk-UK">Я вивчаю CSS3</p>
     <p lang="en-US">I study CSS3</p>
     <p lang="de-DE">Ich lerne CSS3</p>
   </form>
```

```
  </body>
</html>
```

Отримуємо

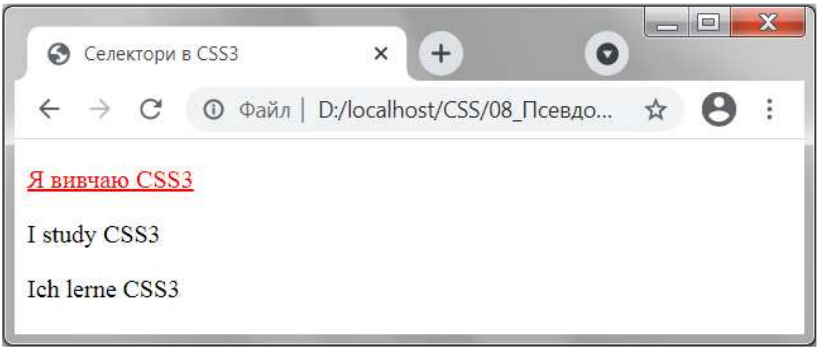


Рис. 3.8. Використання псевдокласу *:lang*

3.1.7. Псевдокласи дочірніх елементів

Особливу групу псевдокласів утворюють **псевдокласи, які дозволяють стилізувати певні дочірні елементи**, тобто виконують пошук елементів на основі їх положення в *html*-документі:

:first-child стилізує перший дочірній елемент в деякому контейнері;

:last-child стилізує останній дочірній елемент в деякому контейнері;

:only-child стилізує елемент, який є єдиним дочірнім – у деякому контейнері;

:only-of-type стилізує елемент, який є єдиним елементом вказаного типу (тега) у деякому контейнері.

Приклад. Використаємо псевдоклас ***:first-child*** для вибору перших посилань у елементах (у даному випадку, у блоках); псевдоклас ***:last-child*** для визначення стилю останніх посилань в елементах; селектор ***:only-child*** для вибору елементів, які є єдиними дочірніми елементами в контейнерах; селектор ***only-of-type*** для виділення єдиного заголовку певного типу:

```
<!DOCTYPE html>
<html>
  <head>
```

```
 <meta charset="utf-8">
 <title>Псевдокласи дочірніх елементів в CSS3</title>
 <style>
   a:first-child{ color: red; text-decoration:underline; }
   a:last-child { color: blue; text-decoration:line-through; }
   p:only-child { color:maroon; font-style:italic; }
   h2:only-of-type { color: #7092BE;
        text-decoration: underline;}
 </style>
</head>
<body>
 <h2>Найцікавіші новинки</h2>
 <h3>Планшети</h3>
 <div>
    <a> Apple iPad Pro 11 </a><br/>
    <a> Samsung Galaxy Tab S6</a><br/>
    <a> Apple iPad Air (2019)) </a><br/>
    <a> Microsoft Surface Pro 7</a>
 </div>
 <h3>Смартфони</h3>
 <div>
    <p>Топ-смартфони 2020</p>
    <a> Samsung Galaxy A50 </a><br/>
    <a> Xiaomi Mi Note 10 </a>
    <br> Huawei P Smart
  </div>
 <div><p>Дані станом на 2020 рік</p></div>
</body>
</html>
```

Стиль по селектору ***a:first-child*** застосовується до посилання, якщо воно є першим дочірнім елементом будь-якого елементу: у першому блоці елемент посилання є першим дочірнім елементом, тому до нього застосовується визначений стиль, а в другому – першим елементом є параграф, тому до жодного посилання цього блоку стиль не застосовується.

Аналогічно, останнім дочірнім елементом у першому блоці є посилання, тому до нього застосовується стиль, визначений селектором ***a:last-child***, у другому – останнім дочірнім елемен-

том є тег переходу на новий рядок, тому стиль не застосовується.

Елемент *p* – єдиний дочірній елемент у третьому блоці *div*, тому до нього застосовується стиль ***p:only-child***.

Заголовок *h2* стилізується селектором ***h2:only-of-type***, оскільки цей заголовок є єдиним елементом *h2* в документі.

Результат застосування вищенаведених стилів:

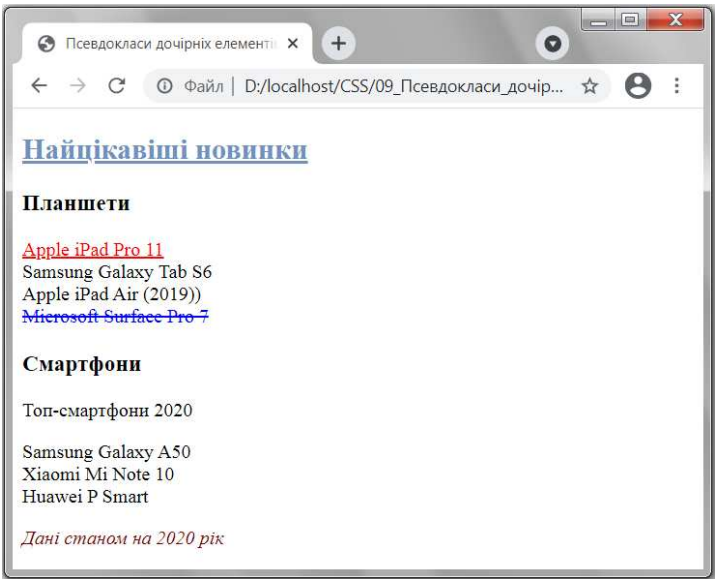


Рис. 3.9. Застосування псевдокласів дочірніх елементів

Псевдокласи, що забезпечують можливість стилізації дочірніх елементів залежно від вказаного параметру:

:nth-child(n) стилізує дочірній елемент під номером *n*, наприклад, другий, третій елемент, тільки парні або тільки непарні елементи і т.д.;

:nth-last-child(n) стилізує дочірній елемент, який має номер *n*, починаючи з кінця;

:nth-of-type(n) стилізує дочірній елемент певного типу під номером *n*;

:nth-last-of-type(n) стилізує дочірній елемент певного типу, що має номер *n*, починаючи з кінця.

При цьому нумерація елементів від 1.

Приклад. Використовуючи псевдоклас ***nth-child(n)***, стилізуємо парні та непарні рядки, вказаний рядок таблиці, а також рядки, порядковий номер яких обчислюються за формулою $a \cdot n+b$, $n=0,1,\ldots$. Застосуємо псевдоклас ***nth-of-type(n)*** до другого рядка та ***nth-last-child(n)*** до передостаннього рядка таблиці:

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>Псевдокласи дочірніх елементів у CSS3</title>
   <style>
    tr:nth-child(odd)    { background-color: yellow; }
    tr:nth-child(even)   { background-color: #bbb; }
    tr:nth-child(1)      { color:red; text-decoration:underline;}
    tr:nth-child(2n+3) { text-transform: uppercase; }
    tr:nth-of-type(2)    { font-style:italic;}
    tr:nth-last-child(2) { font-weight:bold;}
   </style>
 </head>
 <body>
   <h3>ТОП-10 популярних смартфонів
        в Україні 2020</h3>
   <table>
   <tr>
      <td>1</td>
      <td>Apple</td>
      <td>iPhone 11 </td>
   </tr>
   <tr>
      <td>2</td>
      <td>Xiaomi </td>
      <td>Redmi Note 8 Pro</td>
   </tr>
   <tr>
      <td>3</td>
      <td>Samsung</td>
      <td>Galaxy M31</td>
```

```
    </tr>
    <tr>
        <td>4</td>
        <td>Samsung </td>
        <td>Galaxy S10e</td>
    </tr>
    <tr>
        <td>5</td>
        <td>Xiaomi </td>
        <td>Redmi Note 8T</td>
    </tr>
    <tr>
        <td>6</td>
        <td>Xiaomi </td>
        <td>Mi 9T</td>
    </tr>
    <tr>
        <td>7</td>
        <td>Samsung </td>
        <td>Galaxy A51</td>
    </tr>
    <tr>
        <td>8</td>
        <td>Apple</td>
        <td>iPhone Xs</td>
    </tr>
    <tr>
        <td>9</td>
        <td>Samsung</td>
        <td>Galaxy A71</td>
    </tr>
    <tr>
        <td>10</td>
        <td>Apple</td>
        <td>iPhone Xr </td>
    </tr>
    </table>
</body>
```

</html>

Результат застосування стилів:

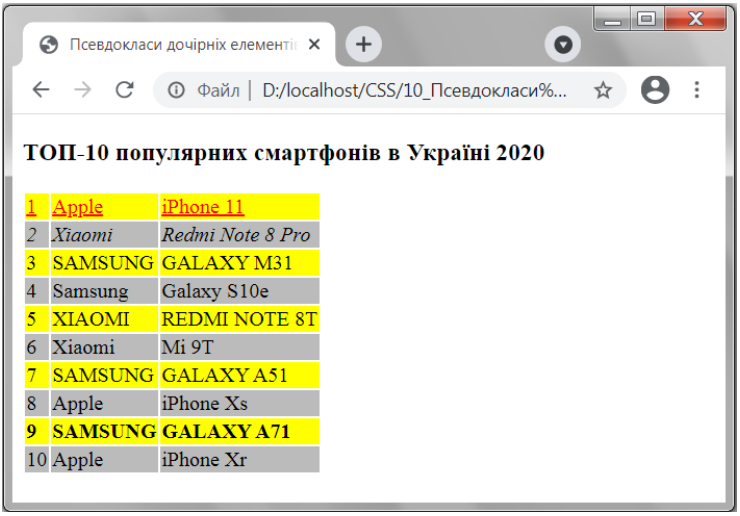


Рис. 3.10. Стилізація дочірніх елементів (рядків таблиці)

Щоб визначити стиль для **непарних елементів**, у селектор передається значення *"odd"*: *tr:nth-child(odd);* для стилізації **парних елементів** у селектор передається значення *"even"*: *tr:nth-child(even).*

Також у цей селектор можна передати номер елемента для стилізації: *tr:nth-child(1)*, у цьому випадку стилізується перший рядок.

Ще одна можливість – використання замінника для номера, який визначається літерою *n*: *tr:nth-child(2n+3).* Цей стиль застосовується до кожного непарного рядка.

Зокрема, рядки, що стилізуються за формулою $a \cdot n+b$, де a,b – числові фіксовані цілочислові параметри, можна визначити, послідовно підставляючи у формулу цілі значення для *n*, починаючи з 0.

Псевдоклас *nth-last-child(n)* фактично надає ту ж саму функціональність, що і *nth-child*, тільки відлік елементів не з початку, а з кінця.

Псевдоклас *nth-of-type(n)* дозволяє вибрати дочірній елемент певного типу за вказаним номером: *tr:nth-of-type(2)*. Аналогічно працює **псевдоклас *nth-last-of-type(n)***, тільки відлік елементів з кінця.

3.1.8. Псевдокласи форм

Ряд псевдокласів використовуються для роботи з елементами форм:

:enabled вибирає елемент, якщо він доступний для вибору (у елемента не встановлено атрибут *disabled*);

:disabled вибирає елемент, якщо він не доступний для вибору (у елемента встановлено атрибут *disabled*);

:checked вибирає елемент, якщо у нього встановлено атрибут *checked* (для прапорців і радіокнопок);

:default вибирає елементи за замовчуванням;

:valid вибирає елемент, якщо його значення проходить валідацію HTML5;

:invalid вибирає елемент, якщо його значення не проходить валідацію;

:in-range вибирає елемент, якщо його значення знаходиться в певному діапазоні (для елементів типу повзунка);

:out-of-range вибирає елемент, якщо його значення не знаходиться в певному діапазоні;

:required вибирає елемент, якщо у нього встановлено атрибут *required*;

:optional вибирає елемент, якщо у нього не встановлено атрибут *required*.

Розглянемо приклад використання псевдокласів *:checked*, *:valid* та *:invalid*, *in-range* та *:out-of-range*.

Псевдоклас *:checked* стилізує елементи форми, в яких встановлено атрибут *checked*. Наприклад, селектор *:checked+span* дозволяє вибрати елемент, сусідній із зазначеним елементом форми.

Псевдокласи *:valid* та *:invalid* стилізують елементи форми в залежності від того, проходять вони валідацію чи ні.

Псевдокласи *:in-range* та *:out-of-range* стилізують елементи форми в залежності від того, чи потрапляє їх значення у певний діапазон. Це найперше стосується елемента <*input*

type="number">, для якого атрибути *min* і *max* задають діапазон, в який повинно потрапляти значення, що вводиться в поле.

Отже, *html*-сторінка та стилізація елементів форми на ній:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
 <title>Псевдокласи форм в CSS3</title>
 <style>
     :checked + span {   color: red;
          font-weight: bold;  /* виділення жирним */  }
     input:invalid { border: 2px solid red;  }
     input:valid { border: 2px solid green;  }
     :in-range { border: 2px solid green; }
     :out-of-range { border: 2px solid red;  }
 </style>
</head>
<body>
 <form>
 <h3>Виберіть технологію</h3>
 <p><input type="checkbox" checked name="html5" />
   <span>HTML5  </span> 
   <input type="checkbox" name="dotnet"/>
   <span>.NET</span> 
   <input type="checkbox" name="java"/>
   <span>Java</span> 
 </p>
 <hr><h3>Вкажіть форму навчання</h3>
 <p> <input type="radio" value="stac"
              checked name="educ"/>
    <span>денна</span>
    <input type="radio" value="zao" name="educ"/>
    <span>заочна</span>
 </p>
 <hr> <p><label for="age">
            Зазначте ваш курс </label>
          <input type="number" min="1" max="5" value="1"
            id="age" name="age"/>
```

```
        </p>
    <hr> <p><input type="text" name="login"
                placeholder="Введіть логін" required />
            <input type="password" name="password"
                placeholder="Введіть пароль" required />
            <input type="submit" value="Увійти" />
        </p>
    </form>
</body>
</html>
```

Таким чином, стилізація елементів форми веб-сторінки має вигляд:

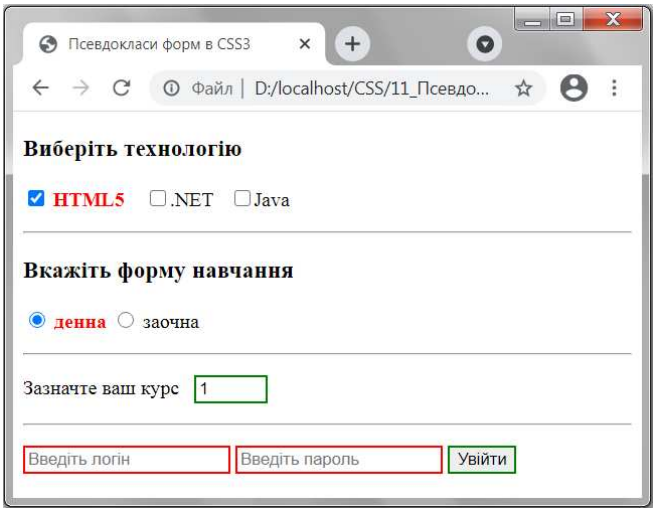


Рис. 3.11. Використання псевдокласів форм

3.1.9. Псевдоелементи

Псевдоелементи надають додаткові можливості з вибору елементів веб-сторінки, їх використання схоже на псевдокласи. В CSS2 псевдоелемент, як і псевдоклас, позначався однією двокрапкою. В CSS3 для відмінності від псевдокласів, **псевдоелементи позначаються двома двокрапками**. Однак для сумісності з більш старими браузерами, які не підтримують CSS3, допустиме використання однієї двокрапки, напр., *:before*.

Перелік найбільш поширених псевдоелементів:

::first-letter стилізує першу літеру в тексті елемента, до якого застосовується;

::first-line стилізує перший рядок тексту елемента, до якого застосовується;

::before додає повідомлення перед певним елементом;

::after додає повідомлення після певного елемента;

::selection застосовується до виділеного користувачем фрагмента тексту.

Приклад. Стилізуємо текст наступним чином: використаємо псевдоелемент *first-letter*, щоб виділити червоним першу літеру тексту; псевдоелемент *first-line* для збільшення розміру першого рядка тексту.

Також застосуємо псевдоелементи *before* і *after* до елементу з класом *warning*. Обидва псевдоелементи мають властивість *content*, яка зберігає текст, що потрібно вставити. Для акцентування уваги використовуємо виділення доданого тексту жирним за допомогою властивості *font-weight: bold*.

Для стилізації, виділеного користувачем тексту, використовуємо псевдоелемент *selection*.

Таким чином, *html*-код сторінки має вигляд:

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>Псевдокласи в CSS3</title>
  <style>
   ::first-letter
      { color:red; font-size: 25px; text-decoration:underline; }
   ::first-line { font-size: 20px; }
   .warning::before
      { content: "Важливо!"; font-weight: bold; }
   .warning::after
      { content: "Будьте обережні!";  font-weight: bold;}
   ::selection { color: white; background-color: black; }
  </style>
 </head>
 <body>
```

```
  <p><span class="warning">
            Висока напруга небезпечно для життя!
      </span>
      <br>Зона високої напруги
   </p>
  </body>
</html>
```

Результат виглядає так:

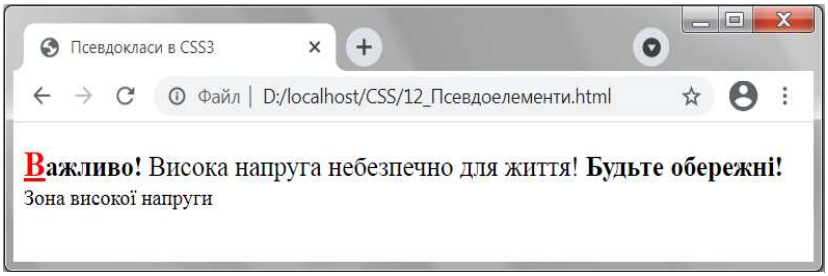


Рис. 3.12. Використання псевдоелементів

3.1.10. Селектори атрибутів

Крім селекторів елементів, можна також використовувати **селектори їх атрибутів**. Наприклад, на веб-сторінці є кілька полів *input*, і потрібно пофарбувати в червоний колір тільки текстові поля. В цьому випадку логічно перевіряти значення атрибуту *type*: якщо воно має значення *“text”*, то це текстове поле, і відповідно, його потрібно пофарбувати в червоний колір. Визначення стилю в цьому випадку виглядає так:

```
input[type="text"]{ border: 2px solid red; }
```

Після елемента, в квадратних дужках, вказується назва атрибуту і його значення. Тобто у цьому випадку для текстового поля встановлюємо стиль – контур червоного кольору товщиною 2 пікселі у вигляді суцільної лінії.

Селектори атрибутів можна застосовувати не тільки до елементів, але і класів та ідентифікаторів. **Наприклад:**

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
```

```
  <title>Селектори атрибутів в CSS3</title>
  <style>
    input[type="text"] { border: 2px solid red; }
    .link[href="http://chnu.cv.ua"] { color: red;
        font-weight: bold; }
  </style>
</head>
<body>
<form align="center" >
    <p><label for="login"> Логін: </label>
    <input type="text" id="login" /></p>
    <p><label for="password">Пароль:</label>
    <input type="password" id="password" /></p>
    <input type="submit" value="Увійти" />
    <p>
      <a class="link"
        href="http://microsoft.com">Microsoft</a> |
      <a class="link" href="https://google.com">Google</a> |
      <a class="link" href="http://chnu.cv.ua">Chnu</a>
    </p>
</form>
</body>
</html>
```

Результат має вигляд:

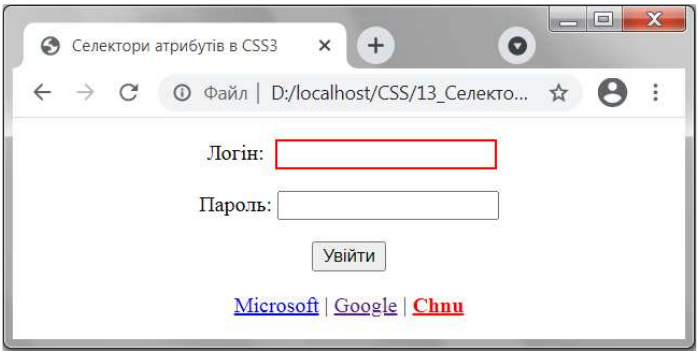


Рис. 3.13. Використання селекторів атрибутів

Спеціальні символи дозволяють конкретизувати значення атрибутів. Зокрема, **символ "^"** дозволяє вибрати всі атрибути, значення яких починаються певним текстом. Наприклад, потрібно вибрати всі посилання, які використовують протокол *https*, тобто посилання повинно починатися з "*https://*". У цьому випадку використовують селектор

a[href^="https://"]{ color: red; }

Якщо значення атрибуту повинно закінчуватися певним текстом, то для перевірки використовують **символ "$"**. Наприклад, потрібно вибрати всі зображення у форматі *jpg*. У цьому випадку, потрібно перевірити, чи закінчується значення атрибуту *src* текстом "*.jpg*":

img[src$=".jpg"]{ width: 100px; }

Символ "*" (зірочка) дозволяє вибрати всі елементи, в яких значення атрибуту містить певний текст, причому не важливо, де саме: на початку, всередині чи в кінці. Наприклад:

a[href="microsoft"]{ color: red; }*

Цей атрибут стилізує всі посилання, які в своїй адресі мають текст "*microsoft*".

3.1.11. Спадкування стилів

HTML-документ являє собою ієрархічне дерево. Це означає, що у кожного елемента (крім кореневого) є тільки один батько, тобто елемент, всередині якого він розташовується.

Наслідування в CSS – це механізм, за допомогою якого значення властивостей батьківського елемента передаються його елементам-нащадкам. Стилі, присвоєні елементу, успадковуються всіма нащадками (вкладеними елементами), якщо вони явно не перевизначені.

Наприклад, нехай на веб-сторінці є заголовок і параграф, які міститимуть текст червоного кольору. Ми можемо окремо до параграфу і заголовку застосувати відповідний стиль, який встановить потрібний колір шрифту:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Наслідування стилів в CSS3</title>
```

```
    <style>
      p {color: red;  text-decoration:underline;}
      h2 {color: red; text-decoration:underline;}
    </style>
  </head>
  <body>
      <h2>Наслідування стилів</h2>
      <p>Текст про наслідування стилів в CSS3</p>
  </body>
</html>
```

Веб-сторінка має вигляд:

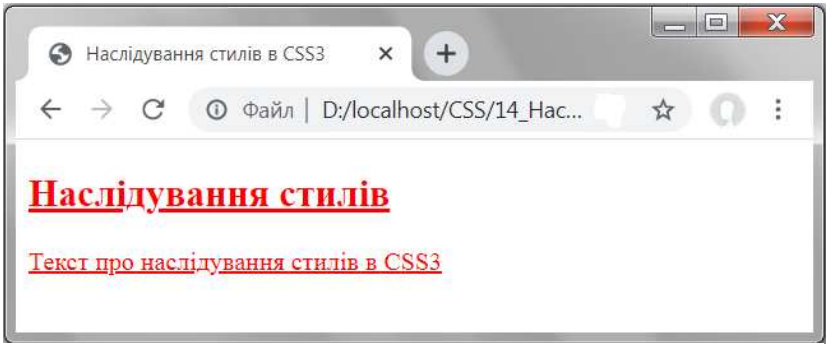


Рис. 3.14. Наслідування стилів

Оскільки і елемент *p*, і елемент *h2* знаходяться в елементі *body*, вони успадковують від цього контейнера – елемента *body* стилі, і щоб не дублювати визначення стилю, можна написати так:

```
<style> body {color: red; text-decoration:underline;}
</style>
```

В результаті визначення стилю стало простішим, а результат залишився тим самим. Якщо успадковувати стиль небажано, то його можна перевизначити для певних елементів:

```
p {color: blue;}
```

Однак **не до всіх властивостей CSS застосовується наслідування стилів**. Наприклад, властивості, які задають відступи (*margin, padding*) і границі (*border*) елементів, не успадковуються.

Крім того, браузери, за замовчуванням, також застосовують ряд попередньо встановлених стилів до елементів. Наприклад, заголовки мають певну висоту і т.д.

3.1.12. Каскадність стилів

Коли до певного елемента застосовується один стиль, то все відносно просто. Однак, якщо ж до одного і того ж елементу застосовується одночасно кілька різних стилів, то виникає запитання: який із цих стилів буде дійсно застосовуватися?

В CSS діє **механізм каскадності**, який являє собою набір правил, що визначають послідовність застосування кількох стилів до одного й того ж елементу.

Якщо до елемента веб-сторінки застосовується кілька стилів, які не конфліктують між собою, то браузер об'єднує їх в один стиль.

Якщо ж стилі конфліктують між собою, наприклад, визначають різний колір тексту, то застосовується **складна система правил для обчислення значущості кожного стилю**. Всі ці правила описані в специфікації по CSS: Calculating a selectors specificity (https://www.w3.org/TR/selectors-3/#specificity).

Розглянемо коротко ці правила.

Для визначення стилю до елемента можуть застосовуватися різні селектори, і **важливість кожного селектора оцінюється в балах**. Чим більше у селектора балів (пунктів), тим він важливіший і тим більший пріоритет його стилі мають над стилями інших селекторів.

- Селектори тегів мають важливість, що оцінюється в **1** бал.
- Селектори класів, атрибутів і псевдокласів оцінюються в **10** балів.
- Селектори ідентифікаторів оцінюються в **100** балів.
- Вбудовані *inline*-стилі (задаються через атрибут *style*) оцінюються в **1000** балів.

Приклад. Нехай стилі визначені наступним чином:

```
<style>
  #index {color: navy;}
     /* темно-синій колір тексту */
  .redLink {color: red; font-size: 20px;}
```

```
    /*червоний колір тексту і висота шрифту 20 пікселів*/
  a {color: black; font-weight: bold;}
    /* чорний колір тексту і виділення жирним */
</style>
```

які одночасно будуть застосовуватися до посилання:

```
<a id="index" class="redLink" href="index.php">
    Основи CSS3
</a>
```

Ці стилі містять два неконфліктуючих правила: *font-size: 20px;* та *font-weight: bold;* які встановлюють висоту шрифту 20 пікселів і виділення посилання жирним. Оскільки кожне із цих правил визначено тільки в одному стилі, то в підсумку вони будуть об'єднані і застосовані одночасно до посилання.

Крім того, всі три стилі містять визначення кольору тексту, але кожен стиль визначає свій колір тексту. Оскільки селектори ідентифікаторів мають більшу вагу, то врешті-решт буде застосований темно-синій колір, що задається селектором: *#index {color: navy;}*.

Якщо селектор складений, то відбувається складання балів всіх вхідних у селектор підселекторів. **Розглянемо на прикладі** визначення стилів для посилань:

```
<style>
  a {font-size: 18px;}
  .nav li a {color: red;}
    /* червоний колір тексту */
  #menu a {color: navy;}
    /* темно-синій колір тексту */
  .nav .menuItem {color: green;}
    /* зелений колір тексту */
  a.menuItem:not(.newsLink) {color: orange;}
    /* оранжевий колір тексту*/
  div ul li a {color: gray; }
    /* сірий колір тексту */
</style>
```

Вище визначено п'ять різних селекторів, які встановлюють колір посилань. В результаті браузер вибере селектор *#menu a* і зафарбує посилання в темно-синій колір. Але на якій підставі браузер вибере цей селектор?

Розглянемо, як підсумовуються бали по кожному з п’яти селекторів:

Селектор	Ідентифікатори	Класи	Теги	Сума
.nav li a	0	1*10	2*1	**12**
#menu a	1*100	0	1*1	**101**
.nav .menuItem	0	2*10	0	**20**
a.menuItem:not(.newsLink)	0	2*10	1	**21**
div ul li a	0	0	4*1	**4**

Отже, для селектора *#menu a* сума виявилася найбільшою – 101 бал. Він містить один ідентифікатор (100 балів) і один тег (1 бал), які в сумі дають 101 бал. У селекторі *.nav .menuItem* два селектори класу, кожен з яких дає 10 балів, тобто в сумі 20 балів. При цьому псевдоклас *:not*, на відміну від інших псевдокласів, не враховується, проте враховується той селектор, який передається у псевдокласі *not*.

Універсальний селектор (*), комбінатори (+, >, ~, ␣), а також псевдоклас *:not* (як згадувалося вище) не впливають на пріоритетність стилів.

CSS надає можливість повністю **скасувати пріоритетність стилів**. Для цього в кінці стилю вказується **ключове слово *!important***:

```
a { font-size: 18px; color: red !important;}
#menu a { color: navy;}
```

У цьому випадку, незалежно від наявності інших селекторів з більшою кількістю балів, до посилань буде застосовано червоний колір тексту, який визначається стилем, що позначений ключовим словом *!important*.

3.2. Основи CSS3. Властивості

3.2.1. Колір в CSS

У CSS є різні властивості, значеннями яких є колір. Досі ми використовували символічні назви, закріплені за найбільш поширеними кольорами, напр. *red*, *blue*, *black* і т.д.

Існує кілька різних способів задання кольору:

1) шістнадцяткове значення вигляду *#RRGGBB*, що складається з окремих компонент: *RR* (*red* – червоний), *GG* (*green* – зелений), *BB* (*blue* – синій), які кодують у шістнадцятковій системі числення значення для червоного (*RR*), зеленого (*GG*) і синього кольорів (*BB*).

Наприклад, *#1C4463*, де перші два символи *1C* – значення червоної компоненти, далі *44* – значення зеленої і *63* – значення синьої складової кольору. Фінальний колір утворюється за допомогою змішування цих значень.

Якщо кожне з трьох двозначних чисел містить по два однакових символи, то їх можна скоротити до одного. Наприклад, *#5522AA* можна скоротити до *#52A;*

2) значення *RGB* також являє собою послідовний набір значень червоної, зеленої і синьої компонент кольору. Значення кожної компоненти кодується числом, що являє собою або процентне співвідношення (0-100%), або число від 0 до 255. Наприклад,

background-color: rgb(100%,100%,100%); / білий колір */*

*background-color: rgb(0%, 0%, 0%); /*чорний колір */*

Між 0 і 100% знаходяться всі інші кольори і відтінки. Як правило, частіше застосовуються значення з діапазону від 0 до 255. Наприклад,

background-color: rgb(28, 68, 99);

3) значення *RGBA* те ж саме, що й значення *RGB* плюс компонента прозорості (*Alpha*). Компонента прозорості має значення від 0 (повністю прозорий) до 1 (непрозорий). Наприклад:

background-color: rgba(28, 68, 99, .6);

4) значення *HSL* (*Hue* – тон, *Saturation* – насиченість і *Lightness* – освітленість). *HSL* задається трьома значеннями. Перше значення *Hue* – кут у крузі відтінків – значення в градусах

від 0 до 360. Наприклад, червоний - 0 (або 360 при повному обороті кола). Кожен колір займає приблизно 51°.

Друге значення *Saturation* представляє насиченість: наскільки чистий колір. Насиченість визначається у відсотках від 0 (повна відсутність насиченості) до 100% (яскравий, насичений колір).

Третє значення *Lightness* визначає освітленість (близькість до білого кольору) і вказується у відсотках від 0 (повністю чорний) до 100 (повністю білий). Для отримання чистого кольору використовується значення 50%.

Наприклад:

background-color: hsl(206, 56%, 25%);

5) значення *HSLA*. Аналогічно *RGBA*, в цьому форматі до *HSL* додається компонента прозорості у вигляді значення від 0 (повністю прозорий) до 1 (непрозорий). Наприклад:

background-color: hsla(206, 56%, 25%, .6);

Також у CSS є спеціальна властивість, яка дозволяє встановити прозорість елементів – **властивість *opacity***, значенням якої є число від 0 (повністю прозорий) до 1 (непрозорий):

```
div{ width: 100px; height: 100px;
    background-color: red; opacity: 0.4; }
```

3.2.2. Стилізація шрифтів. Висота (розмір) шрифту

Властивість *font-family* встановлює набір шрифтів, які будуть використовуватися. Шрифт властивості *font-family* буде працювати, тільки якщо у користувача на локальному комп'ютері є такий шрифт. З цієї причини часто вибираються стандартні шрифти, такі як *Arial*, *Verdana* і т.д., і часто застосовується практика використання кількох шрифтів (задаються через кому). Якщо назва шрифту складається з кількох слів, наприклад *Times New Roman*, то вся назва береться в лапки. Наприклад:

```
body { font-family: Arial; } /*встановлюється шрифт Arial*/
body { font-family: "Times New Roman"; }
body { font-family: Arial, Verdana, Helvetica; }
```

В останньому випадку основним шрифтом є перший – *Arial*, якщо він на комп'ютері користувача не підтримується, то вибирається другий і т.д.

Крім конкретних стилів, також можуть використовуватися загальні універсальні шрифти, що задаються за допомогою значень *sans-serif* і *serif*:

body { font-family: Arial, Verdana, sans-serif; }

Якщо ні *Arial*, ні *Verdana* не підтримуються на комп'ютері користувача, то використовується *sans-serif* – універсальний шрифт без зарубок.

Розглянемо коротко типи шрифтів:

1. Шрифти із зарубками названі так, тому що на кінцях основних штрихів мають невеликі зарубки. Вважається, що вони підходять для великих текстів, оскільки візуально пов'язують одну букву з іншою, роблячи текст більш читабельним. Поширені шрифти із зарубками: *Times New Roman, Georgia, Garamond*. Універсальний узагальнений шрифт із зарубками представляє значення *serif*.

2. Найбільш поширені **шрифти без зарубок**: *Arial, Helvetica, Verdana*.

3. Моноширинний шрифт переважно застосовується для відображення програмного коду і не призначений для виведення стандартного тексту статей. Свою назву ці шрифти отримали від того, що кожна буква в такому шрифті має однакову ширину. Приклади таких шрифтів: *Courier, Courier New, Consolas, Lucida Console*.

Властивість *font-weight* задає товщину шрифту і може приймати 9 числових значень: 100, 200, 300, 400, ... 900, де 100 – дуже тонкий шрифт, 900 – дуже щільний шрифт. Найчастіше для цієї властивості використовують два значення: *normal* (нежирний звичайний текст) і *bold* (напівжирний шрифт):

p { font-weight: normal; }

p { font-weight: bold; }

Властивість *font-style* дозволяє виділити текст курсивом, для цього використовується значення *italic*; щоб скасувати курсив, використовується значення *normal*:

p { font-style: italic; }

p { font-style: normal; }

Властивість *color* встановлює колір шрифту:

p { color: red; }

Властивість *font-size* використовується, щоб задати розмір шрифту.

Розмір шрифту може бути заданий наступними способами:

1. Найбільш поширеною одиницею виміру розміру шрифту є **пікселі**. Щоб задати значення в пікселях, після самого значення вказують скорочення "***px***". Якщо при виведенні тексту на веб-сторінку явно не вказано розмір шрифту, то використовуються значення браузера за замовчуванням. Наприклад, для простого тексту в параграфах це 16 пікселів. Це базовий стиль тексту. Базовий стиль для різних елементів тексту відрізняється: якщо для параграфів це 16 пікселів, то для заголовків рівня *h1* це 32 пікселі, для заголовків *h2* – 24 пікселі і т.д. Наприклад,

div { font-size: 18px; } / розмір шрифту 18 пікселів */*

2. В CSS є **сім ключових слів**, які дозволяють задати розмір шрифту по відношенню до базового: *medium* – базовий розмір шрифту браузера (16 пікселів); *small* – 13 пікселів; *x-small* – 10 пікселів; *xx-small* – 9 пікселів; *large* – 18 пікселів; *x-large* – 24 пікселя; *xx-large* – 32 пікселя. Наприклад:

div{ font-size: x-large; }

3. Відсотки дозволяють задати значення відносно базового або успадкованого шрифту. Наприклад:

div { font-size: 150%; }

У цьому випадку висота шрифту становитиме 150% від базового, тобто *16px * 1,5 = 24px.*

Спадкування шрифту може змінити результуюче значення. Наприклад, стилі визначено наступним чином:

```
<style>
    div { font-size: 10px;}
    p { font-size: 150%;}
</style>
```

і тіло веб-сторінки:

```
<body>
…<div> <p> Lorem ipsum dolor sit amet</p> </div>…
</body>
```

Отже, елемент *p* успадковує від контейнера – блоку *div* шрифт висотою в 10 пікселів, тобто 10 пікселів стають базовими для параграфа. Оскільки для елемента *p* визначається нова висо-

та шрифту в 150%, це означає, що результуюча висота буде *10px * 1,5 = 15px*.

4. Одиниця виміру *em* багато в чому еквівалентна відсоткам. Так, *1em* дорівнює 100%, *.5em* дорівнює 50% і т.д., де % обчислюється від розміру шрифту, заданого в браузері, за замовчуванням, або розміру шрифту батьківського елементу.

3.2.3. Форматування тексту

1. Властивість *text-transform* змінює регістр літер тексту; може набувати таких значень:

- *capitalize*: перетворює першу літеру тексту у верхній регістр;
- *uppercase*: весь текст приводиться до верхнього регістру;
- *lowercase*: весь текст приводиться до нижнього регістру;
- *none*: регістр символів тексту не змінюється, також це значення скасовує зміни регістру літер, внесені цією властивістю.

Наприклад, застосуємо властивість *text-transform* до параграфа:

```
p { text-transform: uppercase;}
```

2. Властивість *text-decoration* дозволяє доповнити текст додатковими ефектами. Ця властивість може набувати таких значень:

- *underline*: підкреслює текст;
- *overline*: надкреслює текст (проводить лінію над текстом);
- *line-through*: закреслює текст;
- *none*: до тексту не застосовується декорування.

Наприклад, застосуємо властивість *text-decoration* до параграфа:

```
p { text-decoration: underline; }
```

Також можна поєднувати різні ефекти властивості *text-decoration* до одного тексту:

```
p { text-decoration: underline overline; }
```

Проілюструємо форматування за допомогою вищенаведених властивостей:

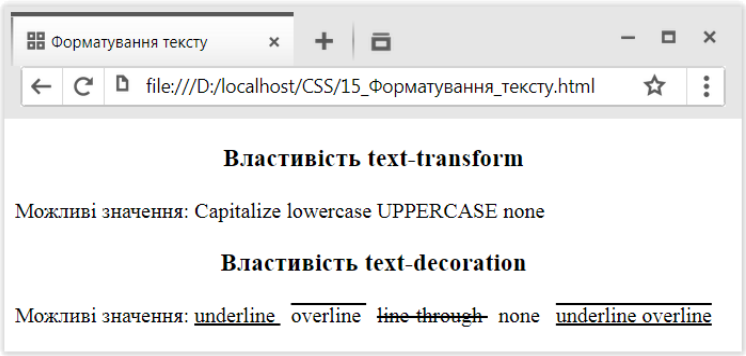


Рис. 3.15. Властивості *text-transform* та *text-decoration*

3. Властивість *text-align* визначає спосіб горизонтального вирівнювання тексту на веб-сторінці. Може набувати таких значень:

- *left*: текст вирівнюється по лівій стороні;
- *right*: текст вирівнюється по правій стороні;
- *justify*: вирівнювання по ширині (слова рівномірно розподіляються по рядку);
- *center*: вирівнювання по центру.

Наприклад, центрування заголовку

h3 { text-align: center; }

4. За допомогою **властивості *text-shadow*** можна створити тінь тексту. Для цієї властивості необхідно задати чотири значення: горизонтальне та вертикальне зміщення тіні відносно тексту, ступінь розмитості тіні і колір створюваної тіні. Наприклад:

h1 { text-shadow: 10px 11px 3px #999; }

h1 { text-shadow: -10px -11px 3px #999; }

У першому випадку горизонтальне зміщення тіні відносно букв становить 10 пікселів вправо, а вертикальне зміщення вниз – 11 пікселів, ступінь розмитості – 3 пікселі, і для тіні використовується колір #999.

Щоб створити горизонтальний зсув вліво, а не вправо, як визначено за замовчуванням, потрібно вказати від'ємне значення, аналогічно для створення вертикального зміщення вгору по-

трібно зазначити також від'ємне значення, як у другому випадку, в прикладі вище.

Таким чином, створені тіні тексту мають вигляд:

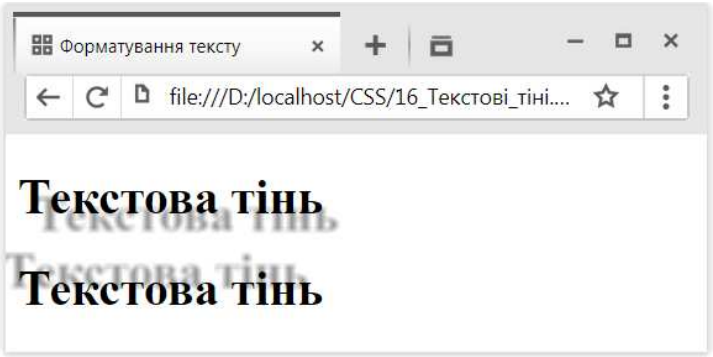


Рис. 3.16. Властивість *text-shadow*: створення тіні елемента

Розглянемо задання відступів та інтервалів:

1. Дві властивості CSS дозволяють управляти **інтервалом між символами і словами тексту**. Для міжсимвольного інтервалу застосовується атрибут *letter-spacing*, а для інтервалу між словами – *word-spacing*:

p.smallLetterSpace { letter-spacing: -1px; }
p.bigLetterSpace { letter-spacing: 1px; }
p.smallWordSpace { word-spacing: -1px; }
p.bigWordSpace { word-spacing: 1px; }

2. Властивість *text-indent* задає відступ першого рядка абзацу. Для задання відступу можуть застосовуватися стандартні одиниці виміру, наприклад *em* або пікселі:

p{ text-indent: 35px; }

3. Властивість *line-height* визначає міжрядковий інтервал, для задання якого можна використовувати пікселі, відсотки або одиниці *em*. Як правило, застосовуються або відсотки, або *em*. Якщо ця властивість не встановлена, то, за замовчуванням, використовується значення *line-height: 120%.*

Наприклад, задамо інтервал для параграфів у документі

p { line-height: 150%; }

3.2.4. Стилізація списків

CSS надає спеціальні властивості для стилізації списків:

1. Однією з таких властивостей є ***list-style-type***. Вона може набувати таких значень **для нумерованих списків:**

- *decimal:* десяткові числа, відлік від 1;
- *decimal-leading-zero:* десяткові числа, в яких одинарні цифри записуються з нулем попереду, напр., *01, 02, 03, ... 99;*
- *lower-roman:* малі латинські цифри, напр., *i, ii, iii, ...;*
- *upper-roman:* великі латинські цифри, напр., *I, II, III, ...;*
- *lower-alpha:* малі латинські літери, напр., *a, b, c ..., z;*
- *upper-alpha:* великі латинські літери, напр., *A, B, C, ..., Z.*

Наприклад: *ol { list-style-type: lower-alpha; }*

Для ненумерованих (маркованих) списків маркерами можуть виступати: *disc* – чорний диск ●; *circle* – порожній кружечок ○; *square* – чорний квадратик ▪.

Наприклад: *ul { list-style-type: disk; }*

Ця властивість може застосовуватися як до всього списку, так і до окремих елементів. Наприклад, визначимо стилі:

```
<style>
    ul.ListDisk {list-style-type: disk;}
    ol.ListLowerAlpha {list-style-type: lower-alpha;}
    .decimal{ list-style-type: decimal; }
    ol.ListLowerRoman{ list-style-type: lower-roman; }
</style>
```

і застосуємо їх до списків на сторінці:

```
<ul class="ListDisk">
   <li>Елемент 1</li>
   <li>Елемент 2</li>
   <li>Елемент 3</li>
</ul>
<ol class="ListLowerAlpha">
   <li>Елемент 1</li>
   <li>Елемент 2</li>
   <li>Елемент 3</li>
</ol>
<ol class="ListLowerRoman">
   <li>Елемент 1</li>
```

*<li class="decimal">Елемент 2
*
*
Елемент 3
*
***
***

Маркований, нумерований та змішаний списки на *html*-сторінці:

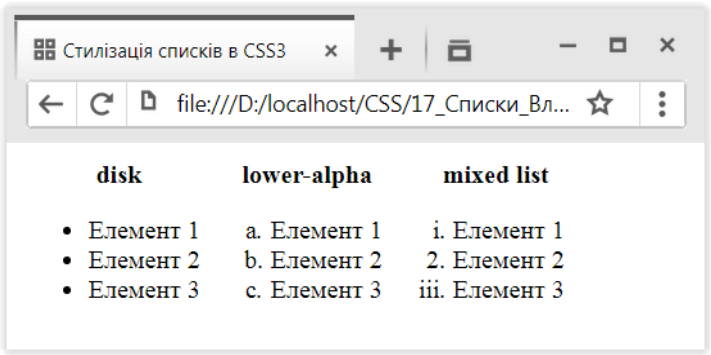


Рис. 3.17. Типи списків в HTML

2. Веб-браузери, зазвичай, відображають маркери зліва від елементів списку. За допомогою **властивості *list-style-position*** можна налаштувати їх позиціонування. Ця властивість приймає два значення: *outside* (за замовчуванням) і *inside* (забезпечує рівномірний розподіл по ширині).

Наприклад, визначимо стилі

<style>
ul.outside { list-style-position: outside; }
ul.inside { list-style-position: inside; }
</style>

і застосуємо їх до списків:

<ul class="inside">
*
... ...*
*
*
<ul class=" outside ">
*
... ...*
*
*

На сторінці ці списки мають такий вигляд:

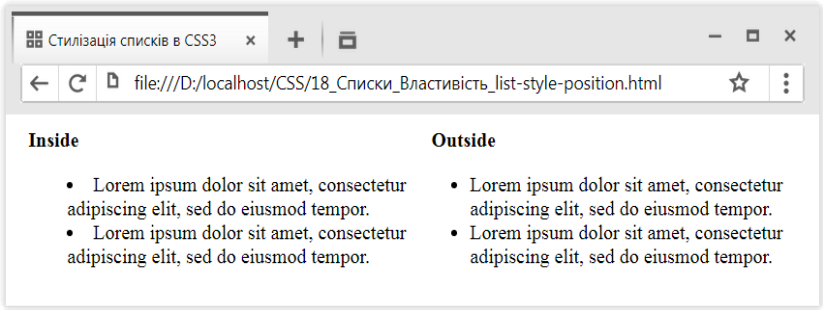


Рис. 3.18. Властивість *list-style-position*

3. Властивість *list-style-image* дозволяє задати маркером списку зображення. Напр.,

ul { list-style-image:url(picture.png); }

Значення параметру *url* – шлях до зображення (*picture.png*), де "*picture.png*" – назва файлу зображення, тобто у цьому випадку мається на увазі, що в одній папці з веб-сторінкою знаходиться файл зображення *picture.png*.

3.2.5. Стилізація таблиць

CSS надає ряд властивостей, які відповідають за стилізацію таблиць:

1. Раніше для задання границі в таблицях широко використовувалась **властивість *border***, наприклад:

<table border="2px">

Зараз спостерігається тенденція використання для стилізації тільки стилів CSS. Тому границя також задається за допомогою CSS, використовуючи властивость *border*:

table { border: 1px solid #ccc; } / границя всієї таблиці */*

tr { border: 1px solid #ccc; } / границя між рядками */*

td, th { border: 1px solid #ccc; } / границя між стовпцями */*

2. При заданні внутрішніх меж за допомогою **властивості *border–collapse*** можна встановити спільну або роздільну межу між суміжними комірками:

- *collapse*: суміжні комірки мають спільну межу;
- *separate*: суміжні комірки мають окремі межі, які розділені простором.

Якщо суміжні комірки мають окремі межі, то за допомогою **властивості *border-spacing*** можна задати розмір простору між ними.

3. Властивість *empty-cells* дозволяє стилізувати порожні комірки, тобто задати режим їх відтворення за допомогою таких значень:

- *show*: порожні комірки відображаються (за замовчуванням);
- *hide:* порожні комірки не відображаються.

Наприклад, якщо у визначенні стилю для таблиці записати

table { empty-cells: hide; }

то порожні комірки разом зі своїми межами (якщо вони задані) не будуть відображатися.

4. Властивість *caption-side* управляє розміщенням заголовку і може набувати таких значень:

- *top:* позиціонування заголовка вгорі (значення за замовчуванням);
- *bottom:* позиціонування заголовка внизу.

Наприклад, *table { caption-side: bottom; }*

5. За допомогою **властивості *table-layout*** задають розмір таблиці. За замовчуванням, ця властивість має значення *auto*, при якому браузер встановлює ширину стовпців таблиці автоматично, виходячи з ширини найширшої комірки у стовпці, а з ширини окремих стовпців складається ширина всієї таблиці. Однак за допомогою значення *fixed* можна встановити фіксовану ширину:

```
table {  border: 1px solid #ccc;  border-spacing: 3px;
         table-layout: fixed;  width:350px; }
```

6. Вертикальне вирівнювання комірок

Як правило, вміст комірок таблиці вирівнюється по центру комірки. Але за допомогою **властивості *vertical-align*** цю поведінку можна перевизначити. Властивість *vertical-align* застосовується тільки до елементів <*th*> та <*td*> і може набувати таких значень:

- *top*: вирівнювання вмісту по верхньому краю комірки;
- *baseline*: вирівнювання першого рядка тексту по верхньому краю;

- *middle*: вирівнювання по центру (значення за замовчуванням);
- *bottom*: вирівнювання по нижньому краю.

Наприклад,

```
td, th{ border: solid 1px #ccc;
        vertical-align: bottom; height: 30px; }
```

Продемонструємо використання вищенаведених властивостей, створивши наступні стилі:

```
<style>
  table { float: left; margin-left:20px; margin-right:20px;
          border: 1px solid black; border-spacing: 3px;
          caption-side: top;
          table-layout: fixed; width:300px; }
  td, th { border: solid 1px red; }
  .style1 { border-collapse: collapse; }
  .style2 { border-collapse: separate; empty-cells: hide; }
</style>
```

та застосувавши їх до таблиць:

```
<table class="style1" >
 <caption>Дані станом на грудень 2020 року
 </caption>
 <tr><th>Компанія</th>
      <th>Модель</th>
      <th>Ціна(грн.)</th>
 </tr>
 <tr><td>Samsung</td>
      <td>Galaxy A50</td>
      <td>7 199,00</td>
 </tr>
 <tr><td>Xiaomi</td>
      <td>Mi Note 10</td>
      <td>8 999,00</td>
 </tr>
 <tr><td>Huawei</td><td>P Smart</td><td></td>
 </tr>
</table>
<table class="style2">
```

```
<caption>Дані станом на грудень 2020 року</caption>
 <tr><th>Компанія</th>
      <th>Модель</th>
      <th>Ціна(грн.)</th>
 </tr>
 <tr><td>Samsung</td>
      <td>Galaxy A50</td>
      <td>7 199,00</td>
 </tr>
 <tr><td>Xiaomi</td>
      <td>Mi Note 10</td>
      <td>8 999,00</td>
 </tr>
 <tr><td>Huawei</td>
      <td>P Smart</td>
      <td></td>
 </tr>
</table>
```

Веб-сторінка з таблицями має вигляд:

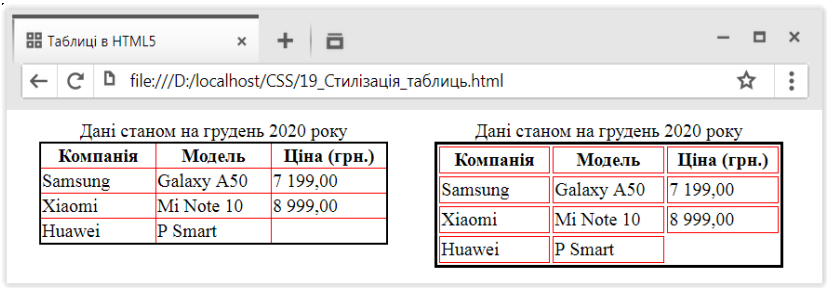


Рис. 3.19. Стилізація таблиць

3.2.6. Блокова модель

Для веб-браузера елементи сторінки являють собою невеликі контейнери або блоки. Такі блоки можуть мати різний вміст – текст, зображення, списки, таблиці та інші елементи. Внутрішні елементи блоків самі виступають блоками.

Схематично блокову модель можна зобразити таким чином:

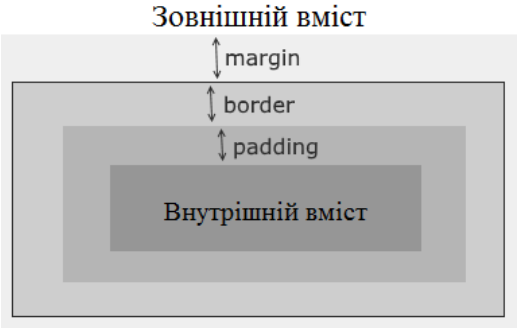


Рис. 3.20. Блокова модель

Нехай елемент розташований у деякому зовнішньому контейнері, напр. в елементі *body*, *div* і так далі. Від інших елементів він відділяється деякою відстанню: зовнішнім відступом, який описується властивістю CSS *margin*. **Властивість *margin*** визначає відстань від границі поточного елемента до інших сусідніх елементів або до границь зовнішнього контейнера.

Далі починається сам елемент, початок якого – це його границя (межа), яка в CSS описується **властивістю *border***.

Після границі елемента розміщується внутрішній відступ, який в CSS описується **властивістю *padding***. Внутрішній відступ визначає відстань від границі елемента до внутрішнього вмісту.

Далі розташовується **внутрішній вміст**, який також реалізує ту ж блокову модель і також може складатися з інших елементів, які мають зовнішні і внутрішні відступи і, відповідно, свої межі.

Наприклад, визначимо наступну веб-сторінку:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Блокова модель в CSS3</title>
    <style>
      div{   margin: 15px; /* зовнішній відступ */
             padding: 11px; /* внутрішній відступ */
             border: 3px solid red;   /*границя шириною
```

```
            три пікселі суцільною червоною лінією*/   }
    </style>
  </head>
  <body>
    <div>
        <p>Перший блок</p>
    </div>
    <div>
        <p>Другий блок</p>
    </div>
  </body>
 </html>
```

Після запуску веб-сторінки у браузері можна побачити блокову модель конкретних елементів. Для цього потрібно натиснути на потрібний елемент правою кнопкою миші і у контекстному меню вибрати пункт, що дозволяє переглянути вихідний код елемента.

Після вибору даного пункту браузер відкриє панель, де буде показано код елемента, його стилі і блокову модель:

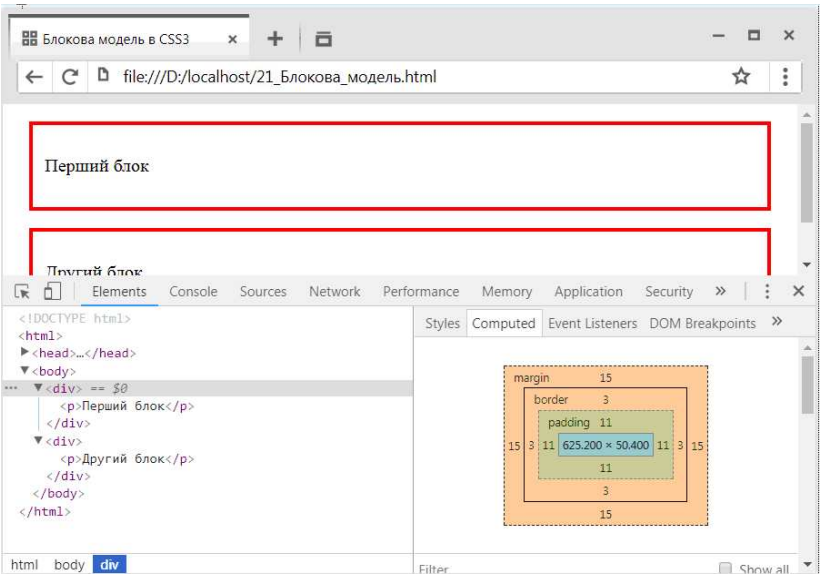


Рис. 3.21. Перегляд у браузері блокової моделі

У цій моделі можна побачити, як задаються відступи елемента, його границя, подивитися відступи від інших елементів і, за необхідності, динамічно змінювати значення їх стилів.

Якщо ми явно не вказуємо значення властивостей *margin*, *padding* і *border*, то браузер застосовує попередньо встановлені значення.

3.2.7. Зовнішні та внутрішні відступи

1. Властивість *margin* визначає зовнішні відступи, тобто відступ елемента від інших елементів або межі зовнішнього контейнера. Існують спеціальні властивості CSS, що задають відступи для кожної сторони:

- *margin-top*: відступ зверху;
- *margin-bottom*: відступ знизу;
- *margin-left*: відступ зліва;
- *margin-right*: відступ справа.

Наприклад, визначимо стиль для селектора *div*:

```
div {   /* зовнішні відступи */
   margin-top: 30px; /* відступ зверху */
   margin-left: 25px; /* відступ зліва */
   margin-right: 20px; /* відступ справа */
   margin-bottom: 15px; /* відступ знизу */
   border: 3px solid red; /* границя */
   }
```

Ці чотири властивості можна записати у вигляді властивості *margin*:

```
div {  margin: 30px 20px 15px 25px;
       border: 3px solid red;
   }
```

Отримуємо блок *div* із визначеними вище зовнішніми відступами (рис. 3.22).

Властивість *margin* задається у форматі:

margin: відступ_зверху відступ_справа відступ_знизу відступ_зліва;

Якщо значення всіх чотирьох відступів збігаються, то можна вказати тільки одне значення: *div { margin: 25px; }*

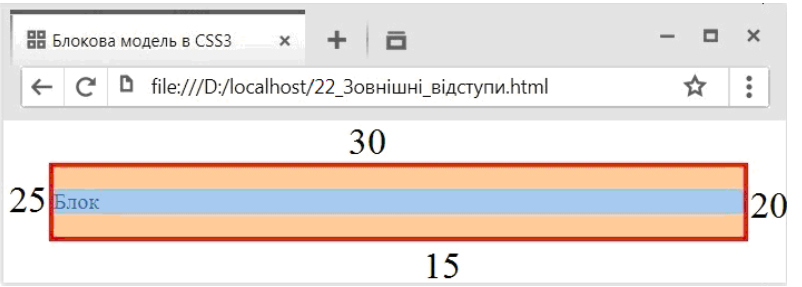


Рис. 3.22. Задання зовнішніх відступів (властивість *margin*)

Для задання відступів можна використовувати **точні значення в пікселях (*px*) або *em***, або **процентні значення**, або **значення *auto*** (автоматичне задання відступів). Наприклад:

margin-left: 2em;

Значення *2em* визначає відстань, яка в два рази більша за розмір шрифта елемента, якщо він визначений. Якщо ж не визначений, то обчислюється залежно від розміру шрифта, що заданий в браузері, за замовчуванням, або розміру шрифта батьківського елемента.

При використанні процентів веб-браузери обчислюють розмір відступів на основі ширини елемента-контейнера, в який вкладено стилізований елемент.

Якщо кілька елементів дотикаються, то браузер вибирає найбільше значення із зовнішніх відступів цих елементів, яке і використовується. Так, у прикладі вище використовувався стиль:

```
div { margin: 30px 20px 15px 25px;
      border: 3px solid red;
    }
```

Розмістимо на веб-сторінці послідовно один за одним блоки *div*:

```
<div> Перший блок </div>
<div> Другий блок </div>
```

Між першим і другим блоками відстань дорівнюватиме 30 пікселів – значення властивості *margin-top* другого блоку *div*, незважаючи на те, що у першого блоку *div* є властивість *margin-bottom*, що дорівнює 15 пікселів.

2. Властивість *padding* задає внутрішні відступи від границі елемента до його внутрішнього вмістимого. Як і для властивості *margin* (зовнішні відступи), в CSS є чотири властивості, які встановлюють відступи для кожної зі сторін:

- *padding-top*: відступ зверху;
- *padding-bottom*: відступ знизу;
- *padding-left*: відступ зліва;
- *padding-right*: відступ справа.

Наприклад, визначимо стиль:

```
div {   margin: 25px;
        padding-top:30px;
        padding-right: 25px;
        padding-bottom: 35px;
        padding-left: 28px;
        border: 2px solid red;
    }
```

Позначимо схематично на рис. задані внутрішні відступи:

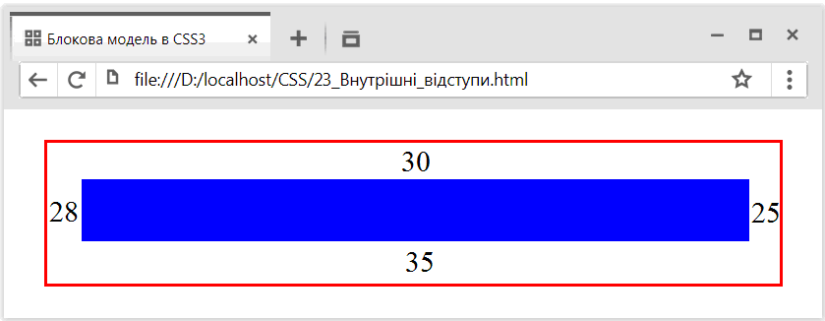


Рис. 3.23. Задання внутрішніх відступів (властивість *padding*)

Для задання значень внутрішніх відступів, як і в *margin*, можуть застосовуватися або конкретні значення в пікселях, або процентні значення (щодо розмірів елементів).

Для задання відступів також можна використовувати скорочений запис:

padding: відступ_зверху відступ_справа відступ_знизу відступ_зліва;

Наприклад:

```
div { margin: 25px;
      padding: 30px 25px 35px 28px;
      border: 2px solid red;
    }
```

Якщо всі чотири значення збігаються, то можна вказати тільки одне значення для всіх відступів:

```
div { margin: 25px;
      padding: 30px;
      border: 2px solid red;
    }
```

3.2.8. Границі

Границя (межа) відділяє елемент від зовнішнього, по відношенню до нього, вмісту. При цьому границя є частиною елемента.

Для налаштувань границі використовуються наступні властивості:

- *border-width* встановлює ширину границі;
- *border-style* задає стиль лінії границі;
- *border-color* встановлює колір границі.

Розглянемо детальніше кожну із цих властивостей:

1. Властивість *border-width* задає ширину границі. Властивість приймає значення в одиницях виміру, тобто *em*, *px* або *cm*, або одне з константних значень: *thin* (тонка межа - 1px), *medium* (середня по ширині - 3px), *thick* (товста - 5px):

```
border-width: 2px;
border-width: medium;
```

2. Властивість *border-color* встановлює колір границі, відповідно, значення кольору CSS:

```
border-color: red;
```

3. Властивість *border-style* задає тип лінії границі і може приймати одне з таких значень:

- *none*: границя відсутня;
- *solid*: у вигляді суцільної лінії;
- *dashed*: штрихова лінія;
- *dotted*: лінія у вигляді послідовності точок;
- *double*: границя у вигляді двох паралельних ліній;

- *groove*: границя має тривимірний ефект;
- *inset*: границя ніби вдавлюється всередину;
- *outset*: аналогічно inset, але границя ніби виступає назовні;
- *ridge*: границя також реалізує тривимірний ефект.

Проілюструємо візуально типи ліній:

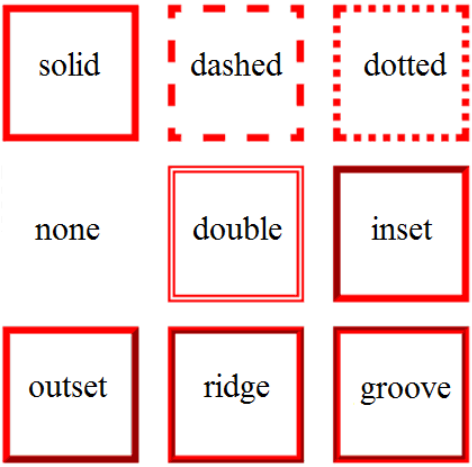


Рис. 3.24. Типи лінії границі

Приклад. Визначимо стиль елемента *div*, в якому границя елемента являє собою червону суцільну лінію товщиною два пікселі:

```
div{   width: 100px; height:100px;
       border-style: solid;
       border-color: red;
       border-width: 2px;
   }
```

При необхідності можна визначити колір, стиль і ширину границі для кожної зі сторін, використовуючи наступні властивості:

- **для верхньої границі:** *border-top-width*, *border-top-style*, *border-top-color*;
- **для нижньої границі:** *border-bottom-width*, *border-bottom-style*, *border-bottom-color*;

- **для лівої границі**: *border-left-width, border-left-style, border-left-color*;
- **для правої границі:** *border-right-width, border-right-style, border-right-color*.

4. Замість задання окремо кольору, стилю і ширини границі можна використати **одну властивість *border***:

border: ширина стиль колір;

Наприклад, *border: 2px solid red;*

Для **задання границі окремо для кожної зі сторін** можна використовувати одну з властивостей: *border-top, border-bottom, border-left, border-right*, їх використання аналогічно:

border-top: 2px solid red;

5. Властивість *border-radius* задає радіус заокруглень всіх кутів границі елемента. Значенням цієї властивості є значення радіуса в пікселях або одиницях *em*. Наприклад,

border-radius: 30px;

У такому випадку кожен кут границі елемента буде заокруглюватися радіусом в 30 пікселів.

Оскільки в елемента може бути максимально чотири кути, то можна вказати чотири значення для задання радіуса кожного з кутів:

border-radius: 15px 30px 5px 40px;

Замість задання радіусів для всіх кутів, можна їх задати окремо. Зокрема, попереднє задання властивості *border-radius* можна записати так:

```
border-top-left-radius: 15px;
   /*радіус для верхнього лівого кута*/
border-top-right-radius: 30px;
   /*радіус для верхнього правого кута*/
border-bottom-right-radius: 5px;
   /*радіус для нижнього правого кута*/
border-bottom-left-radius: 40px;
   /*радіус для нижнього лівого кута*/
```

Також *border-radius* надає можливість створення еліптичних кутів, тобто кут не просто заокруглюється, а використовує два радіуси, утворюючи в результаті контур еліпса:

border-radius: 40px/20px;

У цьому випадку, радіус по осі X має значення 40 пікселів, а по осі Y – 20 пікселів. Схематично це виглядає так:

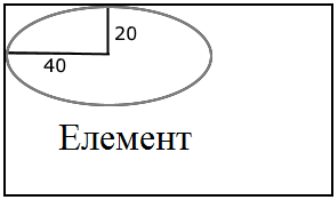


Рис. 3.25. Заокруглення кутів елемента еліптичною дугою

3.2.9. Розміри елементів. Властивість box-sizing

Розміри елементів задаються за допомогою **властивостей *width* (ширина) і *height* (висота)**.

Для цих властивостей значення **за замовчуванням** - ***auto***, тобто браузер сам визначає ширину і висоту елемента. Можна також **явно задати розміри** за допомогою одиниць виміру (пікселів, em) або за допомогою відсотків. Наприклад,

width: 150px;
width: 75%;
height: 15em;

Пікселі визначають точні ширину і висоту.

Одиниця виміру *em* залежить від висоти шрифта в елементі. Якщо розмір шрифта елемента, наприклад, дорівнює 16 пікселів, то *1em* для цього елемента буде дорівнювати 16 пікселів. Тобто якщо у елемента встановити ширину в *15em*, то фактично ширина складе 15·16=230 пікселів. Якщо ж у елемента не визначено розмір шрифту, то він буде взятий з успадкованих параметрів або значень за замовчуванням.

Процентні значення для властивості *width* обчислюються на підставі ширини елемента-контейнера. Якщо, наприклад, ширина елемента *body* на веб-сторінці становить 1000 пікселів, а вкладений в нього елемент <*div*> має ширину 75%, то фактична ширина цього блоку <*div*> становитиме 1000·0.75=750 пікселів. Якщо користувач змінить розмір вікна браузера, то ширина елемента *body*, і відповідно, ширина вкладеного в нього блоку *div*, теж зміниться.

Процентні значення для властивості *height* працюють аналогічно властивості *width*, тільки тепер висота обчислюється по висоті елемента-контейнера.

У той же час **фактичні розміри елемента** можуть в результаті відрізнятися від тих, які встановлені у властивостях *width* і *height*. Розглянемо на прикладі. Нехай стиль визначено наступним чином:

```
div.outer {
  width: 200px;  height: 100px;
  margin: 10px; padding: 10px;
  border: 5px solid #ccc; background-color: #eee;
  }
```

В дійсності **значення властивості *width* - *200px*** визначає тільки ширину внутрішнього вмісту елемента, а під блок самого елемента буде виділено простір, ширина якого дорівнює ширині внутрішнього вмісту (властивість *width*) + внутрішні відступи (властивість *padding*) + ширина границі (властивість *border-width*) + зовнішні відступи (властивість *margin*). Тобто фактична ширина елемента: 200+2·10(*padding*)+2·5(*border*)**=230** пікселів, а разом із зовнішніми відступами складе **250** пікселів.

Схематично це виглядає так:

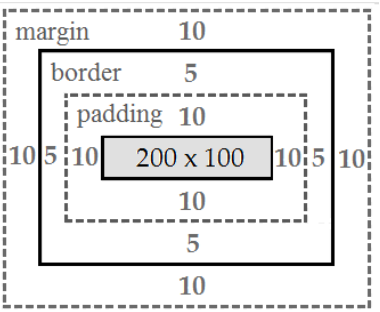


Рис. 3.26. Визначення розмірів елемента

Подібні розрахунки слід враховувати при визначенні розмірів елементів.

За допомогою додаткового набору властивостей можна встановити мінімальні і максимальні розміри:

- *min-width* задає мінімальну ширину;

• *max-width* – максимальну ширину;
• *min-height* – мінімальну висоту;
• *max-height* – максимальну висоту.

Наприклад, визначимо стиль елемента *div*:

```
div { width: 50%;
      min-width: 200px; max-width: 300px; }
```

У цьому випадку, ширина елемента складає 50% ширини елемента-контейнера, але при цьому не може бути менше 200 і більше 300 пікселів.

Перевизначення ширини блоку

Як вже було сказано, блокова модель CSS, за замовчуванням, передбачає, що ширина та висота елемента визначають тільки розміри контенту елемента. Це призводить до того, що розміри потрібно переглядати, якщо якийсь елемент має границю або внутрішні відступи. Наприклад, якщо є чотири блоки з *width: 25%;*, і в якогось із них є межа або внутрішні відступи, то вони вже не помістяться в одному рядку. Для вирішення цього питання призначена властивість *box-sizing*.

Властивість box-sizing визначає, як обчислюються розміри елемента, тобто властивості *width*, *height*, *min-width*, *max-width*, та *min-height*, *max-height*. Надалі, коротко, говоритимемо "розміри" або "ширина та висота елемента".

Властивість приймає одне з наступних значень:

- ***content-box***: значення за замовчуванням, при якому ширина та висота елемента визначають розміри тільки внутрішнього наповнення елемента;
- ***padding-box***: вказує веб-браузеру, що ширина·та висота елемента повинні включати внутрішні відступи як частину свого значення. Наприклад, нехай є такий стиль елемента *div*:

```
div {
   width: 200px; height: 100px;
   margin: 10px; padding: 10px;
   border: 5px solid #ccc; background-color: #eee;
   box-sizing: padding-box;
}
```

Тоді реальна ширина внутрішнього вмістимого блоку дорівнюватиме

200px (*width*) - 10px (*padding-left*) - 10px (*padding-right*) = 180px.

- ***border-box*:** вказує веб-браузеру, що ширина·та висота елемента повинні включати внутрішні відступи і границю як частину свого значення. Наприклад, нехай стиль визначено так:

```
div {
   width: 200px; height: 100px;
   margin: 10px; padding: 10px;
   border: 5px solid #ccc; background-color: #eee;
   box-sizing: border-box;
}
```

Тоді реальна ширина внутрішнього вмісту блоку дорівнюватиме

200px (*width*) - 10px (*padding-left*) - 10px (*padding-right*) - 5px (*border-left-width*) - 5px (*border-right-width*) = 170px.

Отже, якщо виставити елементу ширину 200 пікселів, то ці 200 пікселів будуть включати границі і внутрішні відступи, а контент буде стиснуто, щоб виділити для них місце. Часто такий підхід спрощує роботу з елементами.

Важливо зазначити, що багато сучасних браузерів не підтримують цю властивість.

3.2.10. Фон елемента

Фон елемента описується в CSS ***властивістю background***. Фактично, ця властивість являє собою скорочення набору наступних властивостей CSS:

1. Властивість *background-color* задає колір фону:

```
background-color: #ff0507;
```

2. Властивість *background-image* – як фон використовується зображення:

```
background-image: url(someimage.png); /* в одній папці з вебсторінкою знаходиться файл someimage.png */
background-image: url(http://localhost.com/someimage.png);
  /* використання абсолютної адреси URL */
background-image: url(../images/someimage.png);
  /* використання відносної адреси - шлях відносно
  html-документа*/
```

3. Властивість *background-repeat* встановлює режим повторення фонового зображення по всій поверхні елемента.

CSS належним чином масштабує зображення, щоб найбільш оптимально вписати його у простір елемента. Однак у зв'язку із масштабуванням, зображення може не повністю покривати поверхню елемента. Тому для повного покриття, CSS автоматично починає повторювати зображення.

За допомогою властивості *background-repeat* можна змінити механізм повторення. Вона може набувати таких значень:

- *repeat-x*: повторення по горизонталі;
- *repeat-y*: повторення по вертикалі;
- *repeat*: повторення по обидва боки (за замовчуванням);
- *space*: зображення повторюється стільки разів, щоб повністю заповнити область без створення фрагментів; якщо це не вдається, між картинками додається порожній простір;
- *round*: зображення, належним чином, масштабується для повного заповнення всього простору;
- *no-repeat*: зображення не повторюється.

Наприклад:

```
div {
    width: 200px; height: 150px;
    background-image: url(someimage.png);
    background-repeat: round;
}
```

4. Властивість *background-size* задає розмір фонового зображення по ширині і висоті. Для цього можна використовувати або одиниці виміру, наприклад пікселі, або відсотки, або одне зі значень:

- *contain* масштабує зображення по найбільшій стороні, зберігаючи аспектне відношення (відношення висоти до ширини);
- *cover* масштабує зображення по найменшій стороні, зберігаючи аспектне відношення;
- *auto* означає зображення у повному розмірі (значення за замовчуванням).

Приклади визначення властивості:

```
background-size: cover;
```

background-size: 140px 110px;

Рекомендації для масштабування фонового зображення:

1. Якщо потрібно масштабувати зображення так, щоб оптимально вписати його в фон, то для обох налаштувань можна встановити значення 100%:

background-size: 100% 100%;

2. Якщо задаються точні розміри, то спочатку вказується ширина, а потім висота зображення:

background-size: 200px 150px;

3. Можна задати точне значення для одного виміру – ширини або висоти, а для іншого задати автоматичний розмір, щоб браузер сам обчислив точні значення:

background-size: 200px auto;

5. Властивість *background-position* керує позицією фонового зображення всередині елемента. Вона може приймати відступи від верхнього лівого кута елемента в одиницях виміру, наприклад у пікселях, у такому форматі:

background-position: відступ_по_осі_X відступ_по_осі_Y;

Крім того, ця властивість може приймати одне з таких значень:

- *top:* вирівнювання по верхньому краю елемента;
- *left:* вирівнювання по лівому краю елемента;
- *right:* вирівнювання по правому краю елемента;
- *bottom:* вирівнювання по нижньому краю елемента;
- *center:* зображення розташовується по центру елемента.

Наприклад:

background-position: 20px 15px;

background-position: top right; / зображення вирівнюється по верхньому правому краю, тобто буде розташовуватися у правому верхньому куті елемента */*

6. Властивість *background-attachment* задає, чи буде прокручуватися фонове зображення разом із вмістимим елемента. Ця властивість може набувати таких значень:

- *fixed*: зображення фіксується по відношенню до вікна браузера, тому фон елемента залишається нерухомим при прокручуванні вмісту сторінки і/або вмісту елемента;

• *local*: фонове зображення фіксується по відношенню до вмісту елемента, тобто якщо елемент має прокрутку, фонове зображення прокручується разом із вмістимим елемента;

• *scroll*: фонове зображення фіксується по відношенню до самого елемента і не прокручується разом з його вмістом.

Наприклад:

background-attachment: scroll;

7. Властивість *background-clip* визначає область, яка вирізається із зображення і використовується як фон; може приймає такі значення:

• *border-box*: зображення обрізається по границям елемента;

• *padding-box*: із зображення виключається та частина, яка знаходиться під границями елемента;

• *content-box*: зображення обрізається по вмісту з урахуванням внутрішніх відступів.

Наприклад:

background-clip: padding-box;

8. Властивість *background-origin* вказує позицію, з якої буде починатися фонове зображення для елемента; може набувати таких значень:

• *border-box*: фон елемента встановлюється починаючи з його зовнішньої границі, яка визначається властивістю *border*;

• *padding-box*: фон встановлюється із урахуванням внутрішніх відступів;

• *content-box*: фон встановлюється по вмістимому елемента.

Наприклад: *background-origin: padding-box;*

Властивість *background*, фактично, є скороченням усіх раніше розглянутих властивостей CSS в форматі:

background: <background-image> <background-position>
<background-size> <background-repeat>
<background-origin> <background-clip>
<background-attachment> <background-color>

тобто, якщо є наступний набір властивостей:

background-image: url(image.jpg);
background-repeat: no-repeat;
background-origin: content-box;
background-clip: border-box;

background-attachment: local;
background-color: #eee;
його можна коротко записати так:
background: url(image.jpg) no-repeat content-box
border-box local #eee;

Властивості *background-origin* і *background-clip* мають однакову множину можливих значень. Якщо вказати значення тільки однієї з них, то воно буде застосовано одразу до двох властивостей. Якщо потрібно вказати різні значення для *background-origin* і *background-clip*, то їх записують поруч через пробіл (послідовність важлива – спочатку значення *background-origin*, потім *background-clip*).

3.2.11. Створення тіні елемента

Властивість *box-shadow* дозволяє створити тінь елемента. Ця властивість може приймати кілька значень одночасно (у квадратних дужках вказано необов'язкові параметри):

box-shadow: hoffset voffset [blur] [spread] [color] [inset];

- ***hoffset***: горизонтальне зміщення тіні відносно елемента, при додатному значенні тінь зміщується вправо, а при від'ємному – вліво;
- ***voffset***: вертикальне зміщення тіні відносно елемента, при додатному значенні тінь зміщується вниз, а при від'ємному – вгору;
- ***blur***: необов'язкове значення, що визначає радіус розмиття тіні: чим більше це значення, тим більш розмитими будуть краї тіні, за замовчуванням має значення 0;
- ***spread***: необов'язкове значення, що визначає напрям тіні: додатне значення поширює тінь ззовні у всіх напрямках від елемента, а від'ємне – направляє тінь до елемента;
- ***color***: необов'язкове значення, що задає колір тіні;
- ***inset***: необов'язкове значення, шо визначає тент всередині блоку елемента.

Наприклад, визначимо стиль для блоку *div*:

```
div {  width: 120px; height: 90px;
       margin: 20px; margin-left: 140px;
       border: 3px solid #red;
       box-shadow: 9px 5px 6px 3px #888;
```

}

який, відповідно, буде застосований до кожного елемента *div* на веб-сторінці:

* *

Візуально такий елемент *div* має вигляд:

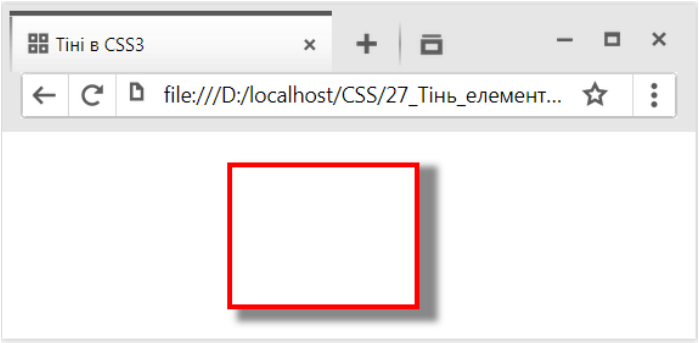


Рис. 3.27. Тінь елемента

Через кому можна визначити кілька різних тіней:

box-shadow: 7px 3px 10px 3px #faa,
11px 4px 11px 3px #888 inset;

3.2.12. Контури елементів

Концепція контурів подібна до використання границь елементів. Але не варто плутати або замінювати границю контуром, тому що вони мають різне значення. **Контур** корисний тим, що дозволяє виділити елемент, щоб привернути до нього увагу в певній ситуації. Контур розташовується поза елементом одразу за його границею.

Контур в CSS3 представлений **властивістю *outline***, яка є скороченням набору властивостей:

1) ***outline-color***: колір контуру;

2) ***outline-offset***: зміщення контуру;

3) ***outline-style***: стиль контуру приймає ті ж значення, що й *border-style*:

- *none*: контур відсутній;
- *solid*: контур у вигляді суцільної лінії;

- *dashed*: штрихова лінія;
- *dotted*: лінія у вигляді послідовності точок;
- *double*: контур у вигляді двох паралельних ліній;
- *inset*: контур ніби вдавлюється всередину;
- *outset*: аналогічно *inset*, але контур ніби виступає назовні;
- *groove*: контур має тривимірний ефект;
- *ridge*: контур також реалізує тривимірний ефект.

4) ***outline-width***: товщина контуру.

Наприклад, визначимо стиль елементів *div* наступним чином:

```
div {  outline-color: red;
       outline-style: dashed;
       outline-width: 3px;
    }
```

За допомогою **властивості *outline*** визначення контуру вище, можна скоротити так:

```
div { outline: red dashed 3px; } /* контур у вигляді червоної
                                    штрих-лінії товщиною 3 пікселі*/
```

Наприклад, елемент *div* стилізований наступним чином:

```
div{  width: 100px; height: 90px;
      margin: 20px; margin-left: 140px;
      border: 5px solid #ccc;
outline: red dashed 5px;
}
```

на веб-сторінці має вигляд:

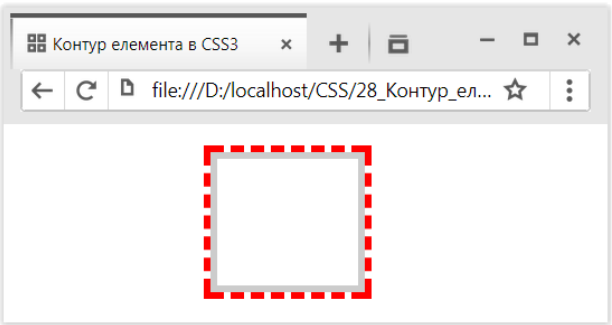


Рис. 3.28. Контур елемента

3.2.13. Обтікання елементів

Як правило, всі блоки та елементи на веб-сторінці розміщуються в тому порядку, в якому вони визначені в коді html. **Для блокових елементів** розташування зверху вниз, оскільки кожний блок займає весь доступний простір по горизонталі, для **рядкових елементів** – зліва направо.

Однак CSS надає спеціальну **властивість *float***, яка дозволяє встановити **обтікання елементів**, завдяки чому можна створити більш цікаві і різноманітні за своїм дизайном веб-сторінки.

Ця властивість приймає одне з наступних значень:

- ***left***: елемент фіксується зліва, а все вмістиме нижче нього, обтікає правий край елемента;
- ***right***: елемент фіксується по правий бік сторінки;
- ***none***: скасовує обтікання і повертає об'єкт у його звичайну позицію.

При використанні властивості *float* до стилізованих елементів, крім елемента *img*, рекомендується встановлювати **властивість *width***.

Елементи, до яких застосовується властивість *float*, ще називають ***плаваючими елементами*** *(*англ. *floating elements)*.

Нехай на веб-сторінці потрібно розмістити зліва від основного тексту зображення, праворуч повинен бути сайдбар, а все інше місце відведено для основного тексту статті.

Визначимо стилі сторінки, застосовуючи властивість *float* наступним чином:

```
<style>
 .image {   float:left;   /* плаваючий елемент зліва */
            margin:10px; margin-top:0px;
}
 .sidebar {  border: 2px solid #ccc;
            background-color: #eee;
            font-size: 20px;
            width: 150px; margin-left:10px;
            padding: 10px;
            float: right;   /* плаваючий елемент справа */
}
</style>
```

```
  </head>
<body>
   <div class="sidebar">Класичний текст
        Lorem Ipsum</div>
   <img src="LoremIpsum.jpg"
        class="image" alt="Lorem Ipsum" />
   <p> Lorem ipsum dolor sit amet, consectetur adipiscing elit,
        sed …</p>
   <p> Duis aute irure dolor in reprehenderit in voluptate
        velit …</p>
  </body>
```

Отримуємо таке розміщення елементів на сторінці:

Рис. 3.29. Обтікання елементів у CSS3

Заборона обтікання. Іноді виникає необхідність заборонити обтікання. Така задача актуальна, якщо деякий блок повинен бути перенесений вниз на новий рядок, а не обтікати плаваючий елемент. Наприклад, футер, як правило, повинен знаходитися строго внизу і розтягуватися по всій ширині сторінки. Якщо ж перед футером знаходиться плаваючий елемент, то футер може обтікати цей елемент, що не коректно.

Для заборони обтікання елементів в CSS застосовується **властивість *clear***, яка вказує браузеру, що стилізований еле-

мент не повинен обтікати плаваючий. Іншими словами, ця властивість визначає, чи стилізований елемент може бути поруч із плаваючим чи розміщений нижче.

Властивість *clear* може приймати наступні значення:

- ***left***: забороняє стилізованому елементу обтікати плаваючий, зафіксований зліва (***float:left***); на обтікання елемента, зафіксованого справа (*float:right*), не впливає;
- ***right***: забороняє стилізованому елементу обтікати плаваючий, зафіксований справа (***float:right***); на обтікання елемента, зафіксованого зліва (*float:left),* не впливає;
- ***both***: стилізований елемент не буде обтікати плаваючі елементи і щодо них зміщується вниз;
- ***none***: стилізований елемент поводить себе стандартним чином, тобто бере участь в обтіканні справа і зліва (значення за замовчуванням).

Розглянемо **приклад веб-сторінки**, де плаваючий елемент – зображення, що фіксується по лівий бік, його обтікають два параграфи, для першого встановимо *float:right*, для другого: *float:left*:

```
  <style>
    .image { float:left;
             margin:10px;
             margin-top:0px; }
    .clearLeft { clear:left; }
    .clearRight { clear:right; }
  </style>
</head>
<body>
    <img src="LoremIpsum.jpg" class="image"
         alt="Lorem Ipsum" />
    <p class="clearRight"> Lorem ipsum dolor sit amet,
         consectetur adipiscing … </p>
    <p class="clearLeft"> Duis aute irure dolor in
         reprehenderit in voluptate....</p>
</body>
```

Тепер розміщення елементів на сторінці виглядає так:

Рис. 3.30. Властивість clear в CSS3

Зазвичай для того, щоб футер не обтікав зображення, а розташовувався нижче, вказують:

.footer { clear: both; …; }

3.2.14. Прокрутка елементів

При створенні веб-сторінок може виникати ситуація, коли вміст блоку займає набагато більше місця, ніж визначено шириною і висотою блоку. У цій ситуації, за замовчуванням, браузер все одно відображає вміст, навіть якщо він виходить за межі блоку.

Властивість *overflow* дозволяє налаштувати прокрутку блоку. Ця властивість може набувати таких значень:

- ***auto***: якщо контент виходить за межі блоку, то створюється прокрутка; в інших випадках смуги прокрутки не відображаються;
- ***hidden***: відображається тільки видима частина контенту; контент, який виходить за межі блоку, не відображається, а смуги прокрутки не створюються;
- ***scroll***: у блоці відображаються смуги прокрутки, навіть якщо весь контент поміщається в межах блоку, і смуг прокрутки не потрібно;

• ***visible***: значення за замовчуванням, контент відображається, навіть якщо він виходить за межі блоку, смуги прокрутки при цьому не створюються.

Приклад. Створимо наступні стилі:

```
<style>
  .article1{  width: 220px;   height: 100px;
      margin:15px;   margin-top:0px;   border: 2px solid #ccc;
      float: left;   overflow: auto;   }
  .article2{  width: 220px;   height: 100px;
      margin:15px;   margin-top:0px;   border: 2px solid #ccc;
      float: left;   overflow: hidden;  }
  .article3{  width: 220px;   height: 100px;
      margin:15px;   margin-top:0px;   border: 2px solid #ccc;
      float: left;   overflow: scroll;  }
  .article4{  width: 220px;   height: 100px;
      margin:15px;   margin-top:0px;   border: 2px solid #ccc;
      float: left;   overflow: visible;  }
</style>
```

Розміщений текст у блокових елементах *div* матиме вигляд:

Рис. 3.31. Прокрутка елементів в CSS3

Властивість *overflow* визначає смуги прокрутки як по вертикалі, так і по горизонталі. За допомогою **додаткових властивостей *overflow-x*** і ***overflow-y*** можна визначити прокрутку, відповідно, по горизонталі і по вертикалі. Ці властивості приймають ті ж значення, що й *overflow*:

overflow-x: auto;
overflow-y: hidden;

3.2.15. Лінійний градієнт

Градієнт реалізує плавний перехід від одного кольору до іншого. В CSS3 є ряд вбудованих градієнтів, які можна використовувати для створення фону елемента.

Градієнт в CSS не визначається якоюсь спеціальною властивістю. Він лише створює значення, яке присвоюється **властивості *background-image***.

Лінійний градієнт поширюється по прямій від одного кінця елемента до іншого, здійснюючи плавний перехід від одного кольору до іншого.

Для створення градієнта потрібно вказати його початок і кілька кольорів (щонайменше два). Наприклад:

background-image: linear-gradient(left, grey,white);

У цьому випадку **початком градієнта** буде лівий край елемента, який позначається значенням *left*. **Кольори градієнта**: сірий (*grey*) і білий (*white*). Тобто, починаючи з лівого краю елемента до правого, буде плавний перехід від сірого кольору до білого.

У використанні градієнтів є один недолік – різноманіття браузерів змушує використовувати **префікс вендора**:

*-webkit- /*Для Google Chrome, Safari, Microsoft Edge,*
Opera від 15 версії і вище/*
-moz- / Для Mozilla Firefox */*
-o- / Для Opera від 15 версії і вище */*

Наприклад, визначимо стиль із градієнтом для елемента *div*:

```
<style>
   div {  width: 290px; height: 120px; margin-left: 80px;
          border: 3px solid red;
          background-image: linear-gradient(left,grey,white);
          background-image: -o-linear-gradient(left,grey,white);
```

```
        background-image:
                -moz-linear-gradient(left,grey,white);
        background-image:
                -webkit-linear-gradient(left,grey,white);
    }
</style>
```

Використання градієнта на прикладі елемента *div* виглядає наступним чином:

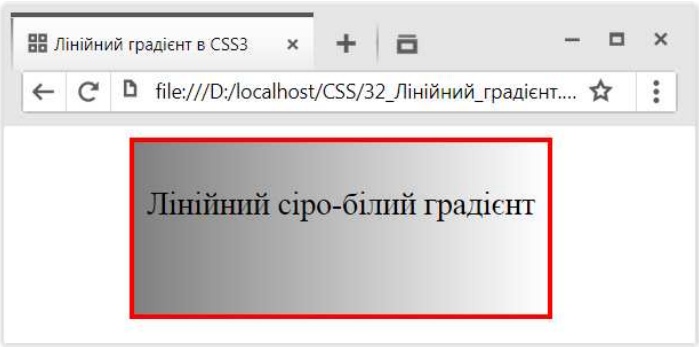


Рис. 3.32. Лінійний графієнт

Для задання початку градієнта можна використовувати наступні значення: ***left*** (зліва направо), ***right*** (справа наліво), ***top*** (зверху вниз) або ***bottom*** (знизу до верху). Наприклад, вертикальний градієнт визначається так:

background-image: linear-gradient(bottom, grey, white);

Також можна задати **діагональний напрямок** за допомогою двох значень:

background-image: linear-gradient(top left, grey, white);

Крім константних значень типу *top* або *left*, також можна **вказати кут від 0 до 360**, який визначить напрямок градієнта:

background-image: linear-gradient(30deg, grey,white);

Після величини кута вказується позначення ***deg***. Наприклад, *0deg* означає, що градієнт починається в лівій частині і переміщається в праву частину, а при вказанні *45deg* він починається в нижньому лівому куті і переміщається під кутом 45° у верхній правий кут.

Після визначення початку градієнта вказують **використовувані кольори або опорні точки**. Кольорів не обов'язково має бути два, їх може бути і більше:

background-image: linear-gradient(top, red, grey, blue);

Усі використовувані кольори розподіляються рівномірно. Однак можна також вказати конкретні місця фону для кольорових точок. Для цього після кольору через пробіл вказується друге значення, яке і визначає положення точки:

background-image: linear-gradient(left, grey, red 30%, red 70%, grey);

За допомогою ***repeating-linear-gradient*** можна створювати повторювані лінійні градієнти. Наприклад:

background-image: repeating-linear-gradient(left, grey 20px, red 30px, #A6E4FF 40px);

У цьому випадку, градієнт починається з лівого краю елемента зі смуги сірого кольору (*grey*) шириною **20** пікселів, далі **30** пікселів перехід до червоного кольору, а потім до **40** пікселів виконується зворотний перехід до блакитного кольору (*#A6E4FF*). Після цього браузер повторює градієнт, поки не заповнить усю поверхню елемента.

3.2.16. Радіальний градієнт

Радіальний градієнт, на відміну від лінійного, поширюється від центру назовні по круговій схемі. Для створення радіального градієнта досить вказати колір, який буде в центрі градієнта, і колір, який повинен бути ззовні. Ці кольори передаються у функцію ***radial-gradient()***. Наприклад:

```
<style>
    div {  width: 150px; height: 150px; margin-left: 100px;
           border: 3px solid red;
    border-radius: 90px;
        /* радіус заокруглення кутів елемента */
    background-image: radial-gradient(white, grey);
    background-image: -moz-radial-gradient(white, grey);
    background-image: -webkit-radial-gradient(white, grey);
    }
</style>
```

Як і у випадку з лінійним градієнтом, потрібно також використовувати **префікси вендорів** для підтримки браузерами.

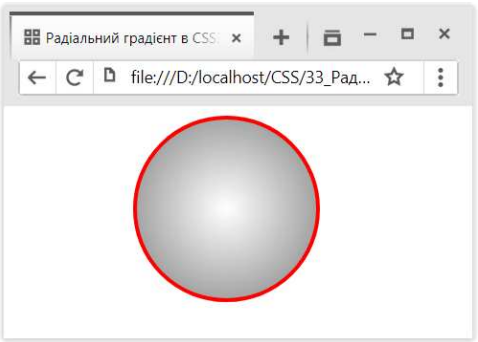


Рис. 3.33. Радіальний графієнт

Радіальний градієнт може мати дві форми: кругову і еліптичну. Еліптична форма являє собою поширення градієнта у вигляді еліпса і задається за допомогою ключового слова ***ellipse***:

background-image: radial-gradient(ellipse, white, grey);

Оскільки це значення для градієнта за замовчуванням, то воно може опускатися при використанні.

Кругова форма являє собою поширення градієнта у вигляді кіл від центру на зовні. Для цього використовується ключове слово ***circle***:

background-image: radial-gradient(circle, white, grey);

Як правило, **центр радіального градієнта** розташований в центрі елемента, але цю поведінку можна перевизначити, вказавши значення параметру *background-position*:

background-position: 10px 20px;
background-image: radial-gradient(circle, white, grey);

У цьому випадку, центр градієнта зміститься на 10 пікселів вправо і 20 пікселів вниз від центра елемента.

Або можна вказати параметри при визначенні градієнта:

background-image: radial-gradient(25% 35%, circle, white, grey);

Числа 25% 35% означають, що центр градієнта буде перебувати на відстані в 25% від лівої межі і в 35% від верхньої межі елемента.

За допомогою спеціальних додаткових значень можна задати **розмір (радіус) градієнта:**

- ***closest-side***: радіус градієнта обчислюється як відстань від центра до, найближчої до центра, сторони елемента, при цьому весь градієнт всередині елемента;
- ***closest-corner***: радіус градієнта обчислюється як відстань від його центра до найближчого кута елемента;
- ***farthest-side***: градієнт поширюється від центра до найвіддаленішої сторони елемента;
- ***farthest-corner***: радіус градієнта обчислюється як відстань від його центра до найбільш віддаленого кута елемента.

Наприклад,

```
background-image:
    radial-gradient(25% 35%, circle farthest-corner,
                    white, grey);
```

3.3. Створення макета сторінки і верстка

3.3.1. Блокова верстка. Частина 1

Як правило, веб-сторінка складається з набору різноманітних елементів, які можуть мати складну структуру. Тому при створенні веб-сторінки виникає необхідність позиціонувати ці елементи, стилізувати їх так, щоб вони розташовувалися на сторінці потрібним чином. Тобто виникає питання **створення макета сторінки, її верстки**.

Існують різні способи, стратегії і види верстки. Спочатку поширеною була **таблична верстка** (на основі html-таблиць), оскільки таблиці дозволяють при необхідності легко і просто розділити весь простір веб-сторінки на рядки і стовпці. Рядками і стовпцями досить просто управляти, в них легко позиціонувати будь-який вміст. Саме це і визначило популярність табличної верстки.

Однак таблична верстка створює не найбільш гнучкі по дизайну сторінки, що особливо актуально в світі, де немає одного розширення екрану, але є великі екрани телевізорів, малі екрани планшетів і фаблетів, досить малі екрани на смартфонах і т.д. Усе це різноманіття екранів таблична верстка виявилася не в змозі задовольнити. Тому поступово їй на зміну прийшла бло-

кова верстка. **Блокова верстка** – це відносно умовна назва способів і прийомів верстки, коли для розмітки використовується CSS-властивість *float*, а основним будівельним елементом веб-сторінок є елемент *div*, тобто, фактично, блок. Використовуючи ці інструменти, можна створити структуру сторінки, яка буде значно гнучкішою, ніж при табличній верстці.

Створимо веб-сторінку з двох колонок, використовуючи властивость *float*. Припустимо, вгорі і внизу будуть, стандартно, шапка і футер, а в центрі – дві колонки: зліва колонка з меню або сайдбар і колонка з основним вмістом справа:

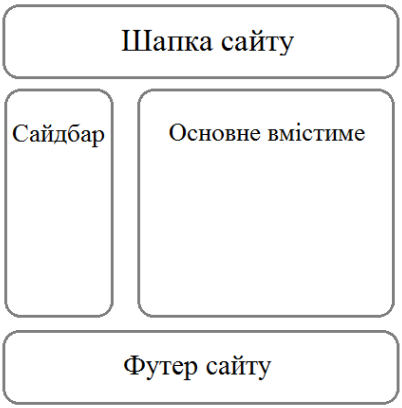


Рис. 3.34. Ескіз створюваної веб-сторінки

При роботі з елементами, які використовують обтікання та властивість *float*, важливий їх порядок. Тому код елемента, для якого встановлюється властивість *float* (**такий елемент називається плаваючим**), повинен розташовуватися перед елементом, який обтікає плаваючий елемент. Тобто блок сайдбару буде розташовуватися перед блоком основного вмісту.

Далі, щоб **перемістити блок сайдбару вліво** по відношенню до блоку основного вмісту і отримати ефект обтікання, потрібно вказати у блоці сайдбару властивість *float: left* і бажану ширину. Ширина може бути фіксованою, наприклад *150px* або *8em*, або можна також використовувати проценти, наприклад,

30% означає 30% від ширини контейнера *body*. З одного боку, блоками з фіксованою шириною легше управляти, а з іншого – процентні значення ширини дозволяють створювати більш гнучкі, гумові блоки, які змінюють розміри при зміні розмірів вікна браузера.

Останнім кроком є **задання відступу блоку основного вмісту від блоку сайдбару**. Оскільки при обтіканні, блок основного вмісту може обтікати плаваючий елемент і праворуч і знизу, якщо плаваючий елемент має меншу висоту, то потрібно встановити відступ, рівний, як мінімум, ширині плаваючого елементу. Наприклад, якщо ширина сайдбару дорівнює **150px**, то для блоку основного вмісту можна задати відступ зліва – **170px** (з яких 150px займає сайдбар, а 20px для двох відступів від сайдбару – зліва і справа), що дозволить створити порожній простір між двома блоками.

При цьому **не варто у блоці основного вмісту явно вказувати ширину**, оскільки браузер розширить її автоматично, щоб основний вміст займав весь доступний простір.

Отже, беручи до уваги всевищесказане, запишемо код сторінки:

```
<html>
 <head>
  <meta charset="utf-8">
  <title> Блокова верстка в HTML5</title>
  <style>
  div { margin: 10px;   border: 1px solid black;
       font-size: 24px;   height: 80px;   }
  #header { background-color: #ccc;    }
  #sidebar { background-color: #ddd;
              float: left;
             width: 150px;  }
  #main { background-color: #eee;
          height: 200px;
          margin-left: 170px;
      /*150px (ширина сайдбару) + 2·10px (2 відступи) */}
  #footer { background-color: #ccc;  }
  </style>
 </head>
```

```
<body>
  <div id="header">Шапка сайту</div>
  <div id="sidebar">Сайдбар</div>
  <div id="main">Основне вмістиме</div>
  <div id="footer">Футер</div>
</body>
</html>
```

Границя і забарвлення блоків, у даному випадку, додані тільки для візуального ефекту. Висота блоків вказана умовно для більшої наочності, в реальності, як правило, висоту буде автоматично встановлювати браузер.

Таким чином, виходить наступна сторінка:

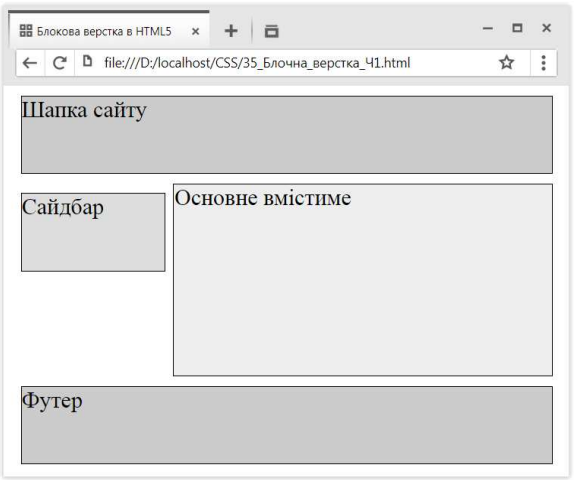


Рис. 3.35. Макет веб-сторінки

Створення правого сайдбару здійснюється аналогічно, тільки потрібно встановити для сайдбару значення *float: right,* а для блоку основного вмісту – відступ справа.

3.3.2. Блокова верстка. Частина 2

Подібним чином можна додати на веб-сторінку більшу кількість колонок і зробити більш складну структуру.

Наприклад, **додамо на веб-сторінку другий сайдбар**. Знову ж таки код обох сайдбарів повинен розташовуватися до блоку з

основним вмістом, який їх обтікає. Змінимо стилі обох сайдбарів і основного блоку:

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="utf-8">
 <title> Блокова верстка в HTML5</title>
 <style>
  div {  margin: 10px;
         border: 1px solid black;
         font-size: 24px;
        height: 80px;    }
  #header { background-color: #ccc; }
  #leftSidebar { background-color: #ddd;
                 float: left;
                width:150px; }
  #rightSidebar { background-color: #ddd;
                  float: right;
                 width:150px; }
  #main { background-color: #eee;
          height: 200px;
          margin-left: 170px;
          margin-right: 170px; }
  #footer { background-color: #ccc; }
 </style>
 </head>
 <body>
  <div id="header">Шапка сайту</div>
  <div id="leftSidebar">Лівий  сайдбар</div>
  <div id="rightSidebar">Правий сайдбар</div>
  <div id="main">Основне вмістиме</div>
  <div id="footer">Футер</div>
 </body>
</html>
```

Тепер веб-сторінка має вигляд:

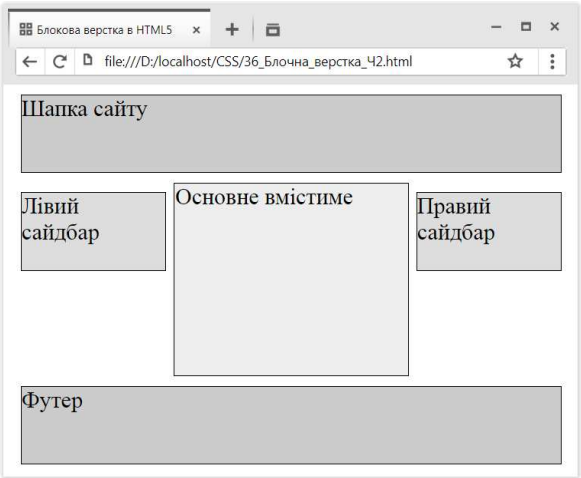


Рис. 3.36. Макет веб-сторінки з двома сайдбарами

3.3.3. Вкладені плаваючі блоки

Можлива ситуація, коли до вкладених елементів **в блоці основного вмісту** також застосовується обтікання. Наприклад, блок основного вмісту може включати блок, власне вмісту і блок меню. Фактично, до таких блоків будуть застосовуватися всі ті ж правила, що були розглянуті раніше.

Розглянемо веб-сторінку, на якій розміщено шапку сайту, футер, основне вмістиме і сайдбар з правого боку. Розмістимо у блоці основного вмісту: блок самого вмістимого та меню – сайдбар зліва.

Схематично, код такої сторінки:

```
<html>
<head>
 <meta charset="utf-8">
 <title> Блокова верстка в HTML5</title>
 <style>
 div { margin: 10px; border: 1px solid black;
       font-size: 24px; height: 80px; }
 #header{ background-color: #ccc; }
 #sidebar{ background-color: #bbb;
           float: right; width: 150px; }
```

```
 #main { background-color:#fafafa;
         height:200px; margin-right:170px; }
  /*170px=150px (ширина сайдбару) +2·10px (2 відступи) */
 #menu { background-color: #ddd;
          float: left;
         width: 160px; }
 #content { background-color: #eee;
            margin-left: 180px; }
 #footer { background-color: #ccc; }
 </style>
</head>
<body>
   <div id="header">Шапка сайту</div>
   <div id="sidebar">Правий сайдбар</div>
   <div id="main">
       <div id="menu">Меню</div>
       <div id="content">Основне вмістиме</div>
   </div>
   <div id="footer">Футер</div>
</body>
</html>
```

Отже, веб-сторінка має вигляд:

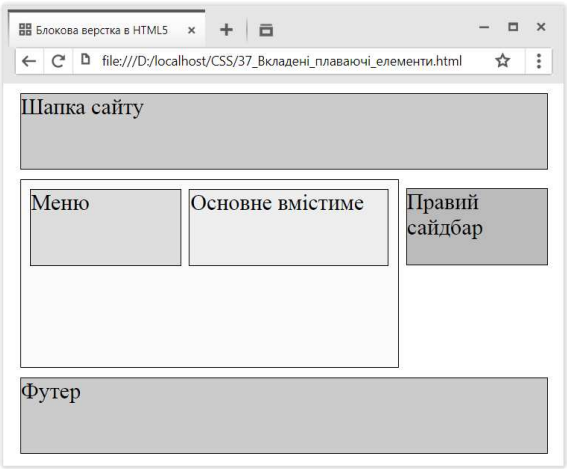


Рис. 3.37. Веб-сторінка з вкладеним плаваючим блоком

3.3.4. Вирівнювання стовпців по висоті

При блоковій верстці може виникнути **проблема висоти стовпців**, яка особливо сильно проявляється, якщо плаваючі блоки мають чітко виражений фон.

Розглянемо це питання на прикладі макету веб-сторінки з попередньої теми (рис. 3.35), наповнивши його деяким вмістом:

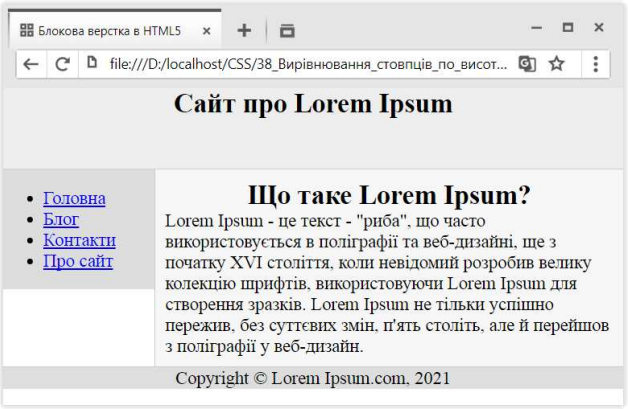


Рис. 3.38. Проблема висоти стовпців

Для цієї сторінки вже визначено обтікання, яке чудово працює, але в залежності від кількості вмістимого одна колонка може бути більша за іншу. В даному випадку (рис. 3.38), якщо в головному блоці багато тексту, то блок меню має недостатню висоту.

Найбільш поширеним **рішенням цієї проблеми** є обгортання плаваючого елементу і блоку, що його обтікає, **в окремий елемент, для якого встановлюється фон**. Потім цей фон використовується найменшим по висоті блоком. В результаті отримуємо ілюзію, що блоки мають рівну висоту і фон у блоків задано коректно.

Доповнимо стилі сторінки новим блоком із зазначенням фону, відповідно, у вкладеному елементі меню фон не вказуємо:

```
<!DOCTYPE html>
 <html>
  <head>
   <meta charset="utf-8">
```

```
<title> Блокова верстка в HTML5 </title>
<style>
body, h2, p { margin:0; padding:0; }
body { font-size: 18px; }
h2 { text-align:center; }
#header { background-color: #eee;
          border-bottom: 1px solid #ccc;
          height: 80px;
        }
#wrapper{ background-color: #ddd; } /* новий блок */
#menu { float: left; width: 150px; }
#main { background-color: #f7f7f7;
        padding: 10px;
        border-left: 1px solid #ccc;
        margin-left: 150px; }
#footer { border-top: 1px solid #ccc;
          background-color: #dedede; text-align:center;
        }
</style>
</head>
```

Знінимо структуру сторінки, додавши в неї новий блок *#wrapper*:

```
<body>
<div id="header"><h2>Сайт про Lorem Ipsum</h2></div>
<div id="wrapper">
  <div id="menu">
   <ul>
     <li><a href="#">Головна</a></li>
     <li><a href="#">Блог</a></li>
     <li><a href="#">Контакти</a></li>
     <li><a href="#">Про сайт</a></li>
   </ul>
  </div>
  <div id="main">
    <h2> Що таке Lorem Ipsum?</h2>
    <p>Lorem Ipsum - це текст - "риба"… </p>
  </div>
</div>
```

```
<div id="footer">
    <p>Copyright © Lorem Ipsum.com, 2021</p>
</div>
</body>
</html>
```

Отримуємо оновлену сторінку:

Рис. 3.39. Вирішення проблеми висоти стовпців

Отже, плаваючий блок *menu* і обтікаючий його блок *main* вкладаються в один елемент *wrapper*, в якому встановлюється фон для елемента *menu*. А елемент *main*, як найбільший елемент, може використовувати свій власний фон.

3.3.5. Властивість display

Властивість display, визначена для елемента, дозволяє керувати його позиціонуванням відносно сусідніх елементів; може набувати наступних значень:

- *inline*: елемент стає рядковим, подібно словам в рядку тексту;
- *block:* елемент стає блоковим, як напр. параграф або *div*;
- *inline-block*: елемент розташовується як рядок тексту;
- *list-item*: елемент позиціонується як елемент списку, зазвичай, з додаванням маркера у вигляді точки або порядкового номера;

• *run-in*: тип блоку елемента залежить від навколишніх елементів;

• *flex*: дозволяє здійснювати гнучке позиціонування елементів;

• *table*, *inline-table*: дозволяє розташувати елементи у вигляді таблиці;

• *none*: елемент не видно і він видалений із розмітки html.

Значення *inline* дозволяє перевизначити елемент в рядковий, якщо елемент таким не є. Відповідно, **значення *block*** дозволяє перевизначити елемент в блоковий. Наприклад, елемент *span*, на відміну від *div*, блоковим не є. За замовчуванням, він має стиль *display:inline*, тому і вбудовується в рядок, а не переноситься на наступний, як параграф або *div*.

Розглянемо приклад перетворення рядкового елемента *span* у блоковий, застосувавши до нього значення *block* властивості *display*, також виконаємо зворотну процедуру – зробимо блоковий елемент *div* рядковим, застосувавши значення *inline* властивості *display*:

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="utf-8">
 <title> Властивість display в CSS3</title>
 <style>
  * { font-size: 18px;}
  p { color: blue; text-align:center; font-weight:bold;}
  span { color: red; background-color: #ccc; }
  .toBlock { display: block; }
  .toLine { background-color: #ccc;  display: inline; }
 </style>
 </head>
<body>
 <p>Властивість display</p>
 <p>значення block</p>
 <div>Це <span> рядковий </span> елемент span</div>
 <div>Це <span class="toBlock"> блоковий </span>
        елемент span</div>
```

```
  <p>значення inline</p>
  <div class="toLine">Рядковий елемент div</div>
</body>
</html>
```

Результат виглядає наступним чином:

Рис. 3.40. Властивість *display:* значення *block* та *inline*

У прикладі вище визначено два елементи *span*, до одного з них застосовується стиль *display: block;*, тому він вважається блоковим, відповідно, виконується перехід на новий рядок.

При перетворенні блокового елемента в рядковий (застосуванні значення *inline*), слід врахувати, що браузер ігнорує деякі властивості елемента, такі як *width*, *height*, *margin*.

Значення *inline-block* представляє елемент, який володіє одночасно ознаками блочного і рядкового елементів. По відношенню до сусідніх зовнішніх елементів, такий елемент розцінюється як рядковий, тобто він не відділяється від сусідніх елементів переходом на новий рядок. Однак по відношенню до вкладених елементів, він розглядається як блоковий, і до такого елемента застосовуються властивості *width*, *height*, *margin*.

Значення *run-in* визначає елемент, який залежить від сусідніх елементів. І тут можливі три випадки:

• елемент оточений блоковими елементами, тоді, фактично, він має стиль *display: block*, тобто сам стає блоковим;

• елемент оточений рядковими елементами, тоді, фактично, він має стиль *display: inline*, тобто стає рядковим;

• у всіх інших випадках, елемент вважається блоковим.

Значення *table*, фактично, перетворює елемент в таблицю. Розглянемо на прикладі списку:

```
<style>
 ul {  display: table;  margin: 0;  }
 li {  list-style-type: none;  display: table-cell;  padding: 10px; }
</style>
```

Список на сторінці:

```
<ul>
   <li>Item 1</li> <li>Item 2</li> <li>Item 3</li>
</ul>
```

Список перетвориться в таблицю, а кожен елемент списку - в окрему комірку. Візуально це виглядає як послідовність із трьох комірок. Замість цього списку ми могли б використовувати стандартну таблицю.

3.3.6. Створення панелі навігації

Фактично, **панель навігації** – це набір посилань, часто у вигляді ненумерованого списку. Панелі навігації бувають найрізноманітнішими: вертикальними і горизонтальними, однорівневими і багаторівневими, але в будь-якому випадку, в центрі кожної навігації знаходиться **елемент <*a*>**. Тому при створенні панелі навігації можуть виникнути ряд труднощів через обмеження елемента посилання. А саме **елемент <*a*> є рядковим**, а це означає, що не можна вказати для нього ширину, висоту, відступи. По ширині посилання автоматично займає те місце, яке йому необхідно.

Нехай, наприклад, **панель навігації на веб-сторінці** визначається наступним чином:

```
<body>
…
 <ul class="nav">
  <li><a href="#">Головна</a></li>
```

```
    <li><a href="#">Блог</a></li>
    <li><a href="#">Контакти</a></li>
    <li><a href="#">Про сайт</a></li>
  </ul>
…
</body>
```

Розглянемо способи стилізації меню та застосуємо їх до панелі навігації, наведеної вище:

1. Створення вертикального меню передбачає перетворення посилання в блоковий елемент:

```
<style>
  ul.nav { list-style: none; }
  ul.nav a {  display: block;  width: 7em;
              padding:10px;
             border: 1px dotted #aaa;
             border-left: 5px solid #aaa;
           }
</style>
```

Стилізована таким чином панель навігації виглядає так:

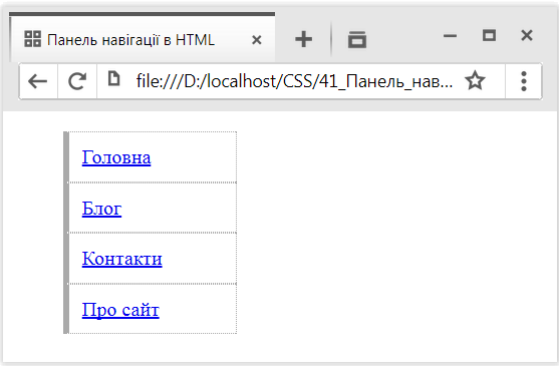


Рис. 3.41. Вертикальна панель навігації

Після задання властивості *display: block* можна визначити для блоку посилання – ширину, відступи і т.д.

2. Для створення горизонтального меню є два методи. Перший полягає у застосуванні властивості *float* і створенні з посилань плаваючих елементів, які обтікають один одного зліва.

Другий метод полягає в створенні рядка посилань за допомогою задання властивості *display: inline-block*.

Перший метод. Алгоритм створення панелі навігації за допомогою **властивості *float*** розділяється на два кроки. На першому кроці для елемента *li*, в якому розміщується посилання, встановлюється *float: left;* що дозволяє розташувати всі елементи списку в ряд при достатній ширині, коли правий елемент списку обтікає лівий елемент списку.

Другий крок полягає в заданні для елемента посилання властивості *display: block*, що дає можливість встановлювати ширину, відступи і всі ті ознаки, які характерні для блокових елементів.

```
<style>
 ul.nav { list-style: none; }
 .nav li { float: left; }
  ul.nav a {  display: block;  width: 5em;  padding:10px;
            margin: 0 5px;  border: 1px dashed #333;
            text-align: center; }
</style>
```

Стилізоване таким чином меню виглядає так:

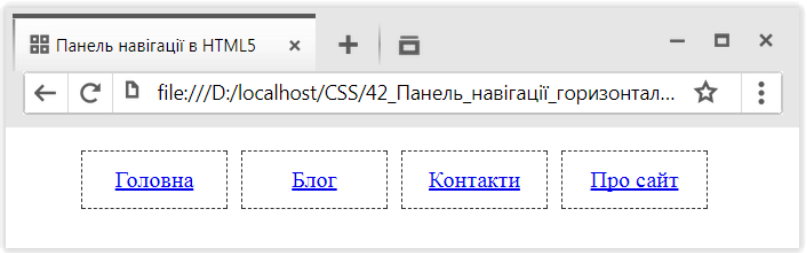


Рис. 3.42. Горизонтальне меню

Другий метод. Інший спосіб створення горизонтальної панелі навігації: зробити кожен елемент ***li*** рядковим, тобто встановити для нього ***display: inline***. Після цього для елемента **посилання**, який розташовується в елементі *li*, встановити ***display: inline-block***:

```
<style>
 ul.nav { list-style: none; }
```

```
.nav li { display: inline; }
ul.nav a { display: inline-block;  width: 5em;
           padding:10px;
           margin: 0 5px;  border: 1px dashed #333;
           text-align: center; }
</style>
```

Стилізоване в такий спосіб меню виглядає так само, як і меню на рис. 3.42.

3.3.7. Вирівнювання плаваючих елементів

Розглянемо основні питання та проблеми, що можуть виникати при використанні плаваючих елементів:

1. Найчастіше може виникати **проблема з виходом вмістимого плаваючих елементів за межі блоків**.

Чому так може відбуватися? Найчастіше браузери своєрідно інтерпретують розміри елемента. Зокрема, у всіх елементів, за замовчуванням, **для властивості *box-sizing*** використовується **значення** *content-box*, тобто при визначенні ширини і висоти елемента, браузер буде додавати до значення властивостей *width* і *height* також і внутрішні відступи padding, і ширину границі. В підсумку, це може призвести до виходу плаваючих елементів з тих блоків, які для них призначені. Тому для всіх елементів рекомендується встановлювати для **властивості *box-sizing*** значення *border-box*, щоб всі елементи вимірювалися однаково, а їх фактична ширина представляла значення властивості *width*. Тому часто в стилях додається наступний стиль:

```
* { box-sizing: border-box; }
```

Значення *box-sizing: border-box;* встановлюється для всіх елементів, і тепер всі вони інтерпретуватимуться браузером однаково.

2. Можлива ситуація, коли на сторінці відображаються незрозумілі відступи. **Проблема відступів** полягає в тому, що браузер, за замовчуванням, визначає для різних елементів вбудовані стилі. Дещо незрозуміло: як і де ці стилі визначені, чому вони застосовуються. Часто для вирішення цієї проблеми розробники “очищають” найбільш значущі стилі для більшості елементів:

html, body, div, span, p, h1, h2, h3, h4, h5, h6, a, img, dl, dt, dd, ol, ul, li, form, table, caption, tr, th, td, article, aside, footer, header { margin:0; padding:0; border:0; font-size:100%; vertical-align:baseline; }

3. Інша проблема – **накладання основного контенту на плаваючий блок навігаційної панелі** – вирішується досить просто – зазначенням для елемента основного вмістимого властивості *clear* зі значенням *both*:

clear: both;

4. Складніша **проблема з виходом плаваючих елементів меню за межі блоку-контейнера**. Є два можливі варіанти рішення. Перше рішення полягає у додаванні до елемента *ul*, який представляє панель навігації, такого стилю:

ul:after { content: " "; display: table; clear: both; }

Друге рішення полягає в тому, щоб зробити сам блок панелі навігації плаваючим:

```
#nav { float: left; width: 100%; clear: both;
       /* інші стильові властивості */}
```

3.3.8. Створення найпростішого макету

Використовуючи отримані з попередніх тем відомості, створимо осмислений макет найпростішої веб-сторінки. Спочатку визначимо базову структуру веб-сторінки:

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <link href="styles.css" rel="stylesheet">
    <title> Базова структура веб-сторінки </title>
 </head>
 <body>
   <div id="header">   <! -- Шапка сайту -->
   <h1>LoremIpsum.com - Сайт про Lorem Ipsum</h1>
     <div id="nav"> <! -- Панель навігації -->
       <ul>
         <li><a href="#">Головна</a></li>
         <li><a href="#">Блог</a></li>
         <li><a href="#">Форум</a></li>
```

```
            <li><a href="#">Контакти</a></li>
            <li><a href="#">Про сайт</a></li>
          </ul>
        </div>
      </div>
    <!–Блок wrapper містить два сайдбари
        та блок основного вмісту-->
    <div class="wrapper">
      <div id="sidebar1" class="aside">
        <h2>Класичний текст Lorem Ipsum, що …</h2>
        <p>"Lorem ipsum dolor sit amet, consectetur ..."</p>
      </div>
      <div id="sidebar2" class="aside">
        <h2>Чому використовується Lorem Ipsum?</h2>
        <p>Давно відомо, що при оцінці дизайну …</p>
        <h3>Опції</h3>
        <ul><li>Опція 1</li>
            <li> Опція 2</li>
            <li> Опція 3</li>
        </ul>
      </div>
      <div id="article">
        <h2>Що таке Lorem Ipsum?</h2>
        <p>Lorem Ipsum - це текст - "риба" ...</p>
        <p>Всупереч поширеній думці, Лорем Іпсум ...</p>
      </div>
      </div>
     <! -- Футер -->
      <div id="footer">
        <p>Contacts: admin@LoremIpsum.com</p>
        <p>Copyright © LoremIpsum.com, 2020</p>
      </div>
     </body>
    </html>
```

На початку розміщена **шапка сайту** – блок із ***header***, який містить заголовок сторінки і панель навігації. Далі **блок *wrapper***, в якому два сайдбари і блок основного вмісту сторінки. Сайдбари, умовно, теж містять деякий вміст, але головне, що

вони визначені до основного блоку. І в кінці невеликий футер.

На початку веб-сторінки визначено підключення файлу ***styles.css***, який буде стилізувати веб-сторінку. Тому створимо в одному каталозі з веб-сторінкою файл *styles.css* і визначимо в ньому наступний вміст:

```
/* 1*/  *{ box-sizing: border-box; }
/* 2*/  html, body, div, span, p, h1, h2, h3, h4, h5, h6, a, ul, li {
          margin: 0;  padding: 0;  border: 0;
          font-size: 100%;  vertical-align: baseline; }
/* 3*/  body { font-family: Verdana, Arial, sans-serif;
          background-color: #f7f7f8; }
/* 4*/  #header { background-color: #f4f4f5; }
/* 5*/  #header h1 { font-size: 24px; text-align:center;
          padding: 30px 30px 30px 10px;  clear: both; }
/* 6*/  #nav { border-top: 2px solid #ccc;
                border-bottom: 2px solid #ccc; }
/* 7*/  #nav li { float: left;  list-style: none; }
/* 8*/  #nav a { display: block;  color: black;
          padding: 10px 25px; text-decoration: none;
          border-right: 1px solid #ccc; }
/* 9*/  #nav li:last-child a { border-right: none; }
/*10*/  #nav a:hover { font-weight: bold; }
/*11*/  #nav:after { content: " ";  display: table;  clear: both; }
/*12*/  .wrapper{ background-color: #f8f7f6; }
/*13*/  .aside h2 { font-size: 0.95em;  margin-top: 15px; }
/*14*/  .aside h3 { font-size: 0.85em;  margin-top: 10px; }
/*15*/  .aside p, .aside li { font-size: .75em;  margin-top: 10px; }
/*16*/  .aside li { list-style-type: none; }
/*17*/  #sidebar1 { float: left;  width: 20%;
                    padding: 0 10px 0 20px; }
/*18*/  #sidebar2 { float: right;  width: 20%;
                    padding: 0 20px 0 10px; }
/*19*/  #article { background-color: #fafafb;
              border-left: 2px solid #cdcdcd;
             border-right: 2px solid #cdcdcd;  margin-left: 20%;
             margin-right: 20%;  padding: 15px;  width: 60%; }
/*20*/  #article:after { clear:both;  display:table;  content:' '; }
/*21*/  #article h2 { font-size: 1.3em;  margin-bottom:15px; }
```

```
/*22*/ #article p { line-height: 150%;  margin-bottom: 15px; }
/*23*/ #footer { border-top: 2px solid #ccc;  font-size: .8em;
        text-align: center;   padding: 10px 10px 30px 10px; }
/*24*/ #nav ul, #header h1, .wrapper, #footer p {
        max-width: 1200px;   margin: 0 auto; }
/*25*/ .wrapper, #nav, #header, #footer { min-width: 768px; }
```

Перші три стилі скидають стильові налаштування, за замовчуванням, для використовуваних нами елементів, а також визначають стиль елемента *body* (детально розглянуто в попередній темі).

Наступна пара **стилів 4-5** управляють відображенням шапки (хедера) і заголовка сторінки.

Набір **стилів 6-11** управляє створенням горизонтальної панелі навігації: для елементів <*li*> встановлюється обтікання (*float: left;*), завдяки чому вони розміщуються в ряд, а кожне посилання робиться блоковим елементом (*display: block;*).

Властивість *content* (стиль 11), використовувана псевдоелементами *before* і *after*, зберігає текст, що потрібно вставити, відповідно, до і після елемента, до якого застосовується псевдоелемент.

Далі налаштування середньої частини сторінки (**стилі 12-18**), зокрема сайдбарів. **Стиль класу *wrapper* (12)** дозволяє встановити фоновий колір для бічних панелей. Для кожного сайдбару визначається ширина в 20% від ширини сторінки. Процентні значення дозволять автоматично підлаштувати ширину блоків під ширину вікна браузера при його розширенні або звуженні.

Далі розміщені стилі **блоку основного вмісту (19-22)** і **футера (23)**. Оскільки бічні панелі мають ширину в 20% кожна, то для головного блока встановлюється ширина в 60% і відступи справа і зліва в 20%.

Стилі 24 і 25 досить важливі.

У стилі 24 для ряду селекторів визначається максимальна ширина в 1200 пікселів. Це означає, що основні елементи сторінки не вийдуть за межі 1200 пікселів, а автоматичний зовнішній відступ зліва і справа дозволить центрувати вміст елементів. Тобто при ширині браузера в 1400 пікселів, ці елементи з шири-

ною 1200 пікселів будуть розміщуватися посередині, а праворуч і ліворуч будуть відступи шириною в (1400-1200)/2 = 100 пікселів.

Стиль 25 дозволяє зробити фіксовану мінімальну ширину для ряду елементів, тобто, в підсумку, при стисканні вікна браузера, сайдбари і основний блок будуть виглядати прийнятно, а при стисканні вікна менше 768 пікселів утворюється смуга прокрутки.

Дана модель розмірів не ідеальна. В подальшому будуть розглянуті більш гнучкі й поширені концепції на основі адаптивної верстки.

В результаті отримуємо простий макет:

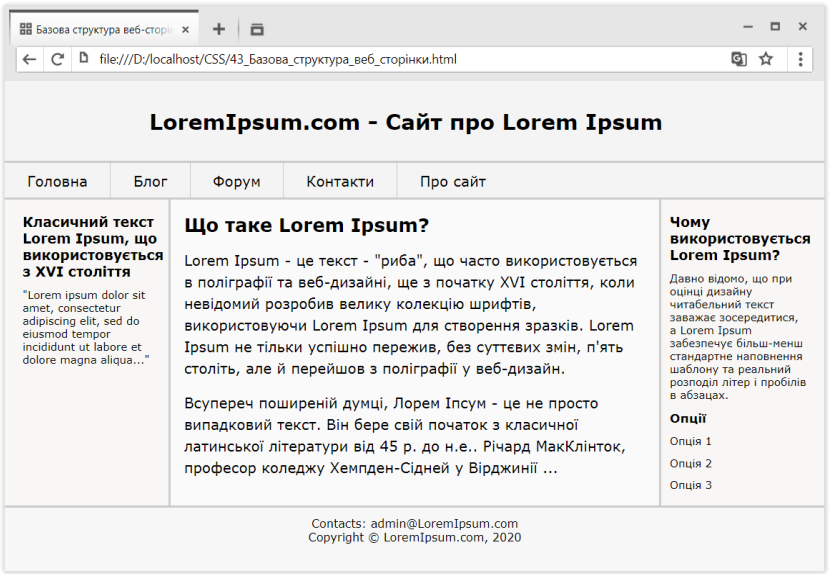


Рис. 3.43. Макет найпростішої веб-сторінки

3.3.9. Позиціонування

CSS надає можливості позиціонування елемента, тобто розміщення елемента у певне місце на сторінці.

Основною властивістю, яка керує позиціонуванням в CSS, є **властивість *position***, яка може приймати одне з наступних значень:

• ***static***: стандартне позиціонування елемента (значення за замовчуванням);

• ***absolute***: елемент позиціонується відносно границь елемента-контейнера, якщо у контейнера властивість *position* не дорівнює *static*;

• ***relative***: елемент позиціонується відносно його позиції за замовчуванням. Як правило, основна мета відносного позиціонування полягає не в тому, щоб перемістити елемент, а в тому, щоб встановити нову точку прив'язки для абсолютного позиціонування, вкладених в нього, елементів;

• ***fixed***: елемент позиціонується відносно вікна браузера, це дозволяє створити фіксовані елементи, які не змінюють положення при прокрутці.

Не слід одночасно застосовувати до елемента властивість *float* і будь-який тип позиціонування, крім *static* (тип за замовчуванням).

Абсолютне позиціонування

Область перегляду браузера можна уявити як прямокутну систему координат, початок відліку якої – точка (0,0) знаходиться у верхньому лівому куті, координатні осі прямують вниз і вправо відповідно. Область має верхній, нижній, правий і лівий краї. Для кожної з цих чотирьох сторін є **відповідна властивість CSS**: ***left*** (відступ від лівого краю), ***right*** (відступ від краю праворуч), ***top*** (відступ від контейнера зверху) і ***bottom*** (відступ знизу). Значення цих властивостей вказуються в пікселях, em або відсотках. Необов'язково задавати значення для всіх чотирьох сторін. Як правило, встановлюють тільки два значення – відступ від верхнього краю *top* і відступ від лівого краю *left*.

Наприклад, розмістимо елемент *div* з абсолютним позиціонуванням на 100 пікселів вліво від межі області перегляду і на 50 від верхнього краю:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
   <title> Абсолютне позиціонування</title>
   <style>
```

```
    div {
      position: absolute; left: 100px; top: 50px;
      width: 75px; height: 75px; background-color: yellow;
    }
  </style>
 </head>
 <body>
    <div></div>
    <p>The standard Lorem Ipsum passage,
        used since the 1500s</p>
    <p>"Lorem ipsum dolor sit amet, consectetur …."</p>
 </body>
</html>
```

Сторінка виглядає наступним чином:

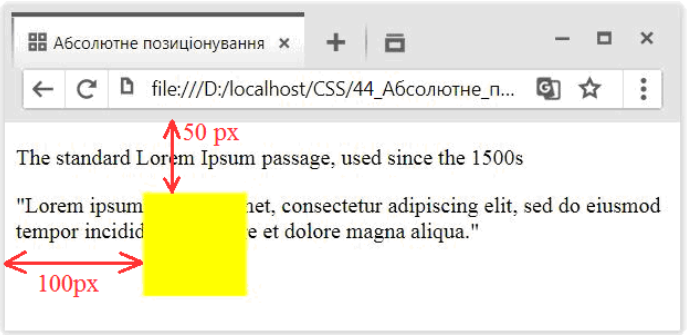


Рис. 3.44. Абсолютне позиціонування елемента

Абсолютне позиціонування елемента на веб-сторінці означає, що елемент буде вилучений зі звичайного потоку документа і розміщений в точному місці на сторінці. Такий елемент не впливає на те, як елементи перед ним або після нього розміщуються на веб-сторінці.

Таким чином, не важливо, що після елемента *div* розміщуються інші елементи. Даний блок *div*, у будь-якому випадку, буде позиціонуватися відносно границь області перегляду браузера.

Зауважимо, якщо елемент з абсолютним позиціонуванням розташовується в іншому контейнері, у якого, в свою чергу,

значення властивості *position* не дорівнює *static*, то елемент позиціонується відносно меж свого батьківського контейнера.

Відносне позиціонування задається за допомогою **значення *relative***. Для задання конкретної позиції, на яку зсувається елемент, застосовуються ті ж властивості *top*, *left*, *right*, *bottom*, але при цьому точкою відліку для обчислення нової позиції елемента є його позиція за замовчуванням. Особливістю відносного позиціонування є те, що простір, який елемент займав би за замовчуванням, залишається порожнім.

Наприклад, якщо в попередньому прикладі змінити значення *absolute* властивісті *position* на *relative*, отримуємо

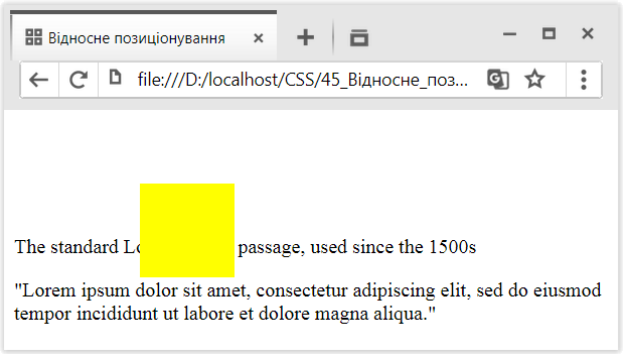


Рис. 3.45. Відносне позиціонування елемента

Фіксоване позиціонування є поширеним способом фіксації в області перегляду браузера деяких елементів. Досить часто на різних сайтах можна побачити фіксовану панель навігації, яка не змінює свого положення незалежно від прокрутки.

Для фіксованого позиціонування в елемента потрібно встановити значення ***fixed* властивості *position***. Після цього за допомогою стандартних властивостей *left*, *right*, *top* і *bottom* можна задати конкретну позицію фіксованого елемента.

Елементи з фіксованим, як і з абсолютним позиціонуванням **не беруть участь у стандартному потоці html** і жодним чином на нього не впливають. Тому потрібно належним чином розмістити елементи стандартного потоку html-розмітки відносно елементів з фіксованим та абсолютним позиціонуванням, напри-

клад, встановивши потрібний відступ від межі області перегляду браузера.

Властивість z-index

За замовчуванням, при накладанні двох позиційованих елементів (таких, для яких **властивість *position*** має значення ***absolute***, ***relative*** або ***fixed***) поверх іншого відображається той елемент, який визначений в html-розмітці останнім.

Накладання елементів дозволяє імітувати третій вимір (вісь Z, яка перпендикулярна екрану). **Властивість *z-index*** дозволяє змінити порядок накладання таких елементів, тобто задати, який елемент буде розташовуватися "вище", а який – "нижче". Значення властивості – число, напр.:

z-index: 100;

Елементи з більшим значенням цієї властивості будуть відображатися поверх елементів із меншим значенням *z-index*.

Дія властивості *z-index* поширюється як на сам елемент, так і на його дочірні елементи, тобто піднімаючи батьківський елемент вище по осі Z, ви піднімаєте і його дочірні елементи.

3.4. Трансформації, переходи і анімації

3.4.1. Трансформації

Одним із нововведень CSS3, у порівнянні з попередньою версією, є можливість здійснювати **трансформації елемента**. До трансформацій відносяться такі дії, як поворот елемента, його масштабування, нахил, переміщення по вертикалі або горизонталі тощо. Для створення трансформацій в CSS3 призначена **властивість *transform***.

Властивость *transform* приймає значення – **"трансформуючі" функції** (англ. *transform-functions*). Ці функції задають тип трансформації (перетворення), а їх параметри визначають ступінь трансформації.

Розглянемо основні типи перетворень:

1. Для повороту елемента властивість *transform* використовує функцію *rotate*:

transform: rotate(angle deg);

де параметр *angle* задає величину кута повороту в градусах. Кут повороту може задаватися додатним і від'ємним значенням.

У випадку від'ємного значення, поворот виконується у протилежну сторону (вліво).

2. Масштабування виконується за допомогою функції *scale*:

transform: scale(k);

де *k* – величина масштабування, значення більше 1, задає розтягування по вертикалі та горизонталі, а менше 1 – стискання, тобто значення 0.5 задає зменшення елемента в два рази, а значення 1.5 – збільшення в півтора раза.

Можна задати величину масштабування окремо по вертикалі і горизонталі. Наприклад, масштабування по горизонталі в 2 рази, а по вертикалі – в 0.5 раза:

transform: scale(2, 0.5);

Також можна окремо визначити параметри масштабування: функція *scaleX()* задає зміну по горизонталі, а *scaleY()* – по вертикалі:

transform: scaleX(2);

transform: scaleY(0.5);

Використовуючи від'ємні значення, можна створити ефект дзеркального відображення, напр.:

transform: scaleX(-1);

3. Для переміщення (зсуву) елемента використовується функція *translate*:

transform: translate(offset_X, offset_Y);

де параметр *offset_X* вказує, наскільки елемент зміщується по горизонталі, а *offset_Y* – по вертикалі. Наприклад, змістимо блок на 30 пікселів вниз і на 50 пікселів вправо:

transform: translate(50px, 30px);

Одиницями виміру зсуву можуть бути не тільки пікселі, але і будь-які інші одиниці вимірювання довжини в CSS – em, % і т.д.

За допомогою додаткових функцій, можна окремо виконати зсув по горизонталі або вертикалі: *translateX()* (по горизонталі) і *translateY()* (по вертикалі), наприклад:

transform: translateX(30px);

Крім додатних значень, також можна використовувати і від'ємні – вони зсувають елемент у протилежну сторону (вліво):

transform: translateY(-2.5em);

4. Для нахилу (перекосу) елемента використовується функція *skew()*:

transform: skew(angleX, angleY);

де перший параметр визначає на скільки градусів нахилити елемент по осі X, а другий - значення нахилу по осі Y, напр.:

transform: skew(30deg, 10deg);

Для створення нахилу тільки по одній осі для іншої осі треба використати значення 0:

transform: skew(45deg, 0); / нахил на 45 градусів по осі X */*

transform: skew(0,45deg); / нахил на 45 градусів по осі Y */*

Для створення нахилу окремо по осі X і по осі Y в CSS є спеціальні функції: *skewX()* і *skewY()*, відповідно:

transform: skewX(45deg);

Також можна передавати від'ємні значення, тоді нахил буде здійснюватися у протилежну сторону (вліво):

transform: skewX(-30deg);

Приклад. Визначимо стилі трансформації елемента:

```
/* 1 */  .rotated { transform: rotate(30deg); }
            /* поворот на 30° */
/* 2 */  .doubleScale { transform: scale(1.5); }
           /* масштабування в 1,5 разів*/
/* 3 */  .halfScale { transform: scale(0.5); }
            /* масштабування в 0,5 рази */
/* 4 */  .translated { transform: translate(70px, 30px); }
            /* зсув */
/* 5 */  .skewed { transform: skew(30deg, 10deg); }
            /* нахил */
div { /* стиль елемента div */
   width: 70px;  height: 70px;
   margin:25px; padding: 40px 10px;
   box-sizing: border-box; display: inline-block;
   background-color: #D4D4D4;
}
```

Застосуємо ці перетворення до набору елементів *div* з однаковими розмірами 70px×70px. Ті елементи, до яких застосовано перетворення, виділені пунктиром і пронумеровані згідно із застосуванням до них стилів, решта елементів наведено для порівняння:

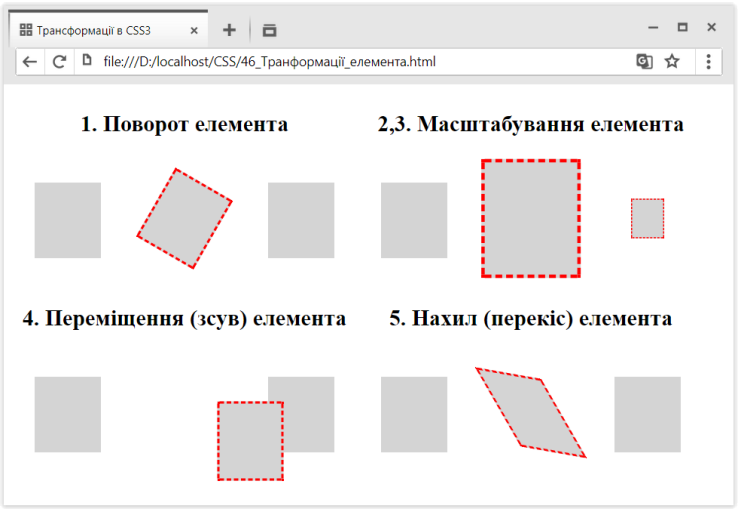


Рис. 3.46. Трансформації елемента

Зауважимо, що при перетворенні елемент може накладатися на сусідні елементи, тому що спочатку відбувається задання положення елемента і тільки потім виконується перетворення.

Комбінування перетворень. Якщо потрібно застосувати до елемента одночасно кілька перетворень, напр. поворот і зсув, то можна їх комбінувати. Наприклад, застосуємо всі чотири перетворення:

```
transform: translate(100px, 50px) skew(30deg, 30deg)
           scale(2.5) rotate(90deg);
```

Браузер застосовує всі ці функції в порядку їх розташування, тобто в цьому випадку спочатку до елемента застосовується зсув, потім нахил, потім масштабування і в кінці поворот.

Вихідна точка трансформації. За замовчуванням, у трансформаціях браузер використовує **центр елемента** як точку початку перетворення. Але за допомогою **властивості *transform-origin*** вихідну точку можна перевизначити. Ця властивість приймає параметр – значення в пікселях, em і відсотках. Також для задання точки можна використовувати ключові слова:

- *left top*: лівий верхній кут елемента;
- *left bottom*: лівий нижній кут елемента;
- *right top*: правий верхній кут елемента;

• *right bottom*: правий нижній кут елемента.

Наприклад, визначимо набір стилів:

```
   /* поворот на 45° вліво */
.transform1{ transform: rotate(-45deg);
  }
   /* поворот на 45° вліво відносно
      лівого верхнього кута елемента */
.transform2{ transform-origin: left top;
             transform: rotate(-45deg);
  }
   /* поворот на 45° вліво відносно
      правого нижнього кута елемента */
.transform3{ transform-origin: right bottom;
             transform: rotate(-45deg);
  }
 div{  width: 100px;  height: 100px;
       margin: 80px 30px;
       box-sizing: border-box;
       float: left;
      background-color: #ccc;
      border: 2px solid black;
  }
```

Результат застосування, визначених вище стилів, до елементів *div* з однаковими розмірами 100px×100px має вигляд:

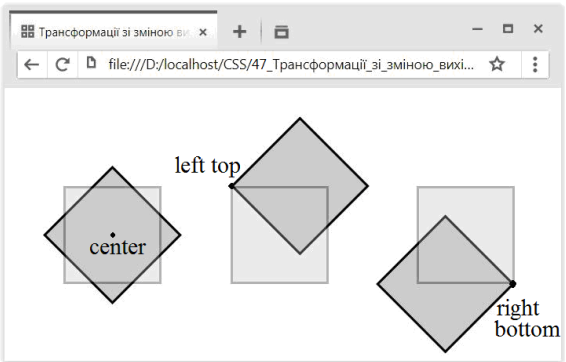


Рис. 3.47. Трансформації зі зміною вихідної точки

3.4.2. Переходи

Перехід (англ. *transition*) являє собою анімацію від одного стилю до іншого протягом певного періоду часу.

Для створення переходу потрібно два набори властивостей CSS: **початковий стиль** для елемента на початку переходу і **кінцевий стиль** – результат переходу.

Розглянемо найпростіший приклад переходу. Визначимо стилі для елемента *div*:

```
<style>
 div {  width: 100px; height: 100px;
        margin: 30px 30px;
        background-color: grey;
        transition-property: background-color;
        transition-duration: 2s; }
 div:hover { background-color: red; }
</style>
```

та тіло веб-сторінки з наявним блоком *div*:

```
<body>  <div></div>  </body>
```

У прикладі вище анімується властивість *background-color* елемента *div*. При наведенні покажчика миші на елемент, він буде змінювати колір із сірого на червоний, а якщо відвести покажчик миші з простору елемента буде повертатися вихідний колір.

Етапи створення анімації стилів:

1. Щоб вказати властивість як таку, шо анімується, її назва передається **властивості *transition-property***:

transition-property: background-color;

2. Далі потрібно задати тривалість переходу в секундах за допомогою **властивості *transition-duration***:

transition-duration: 2s;

Крім секунд, можна встановлювати значення в мілісекундах, наприклад 500 мілісекунд:

transition-duration: 500ms;

Рекомендується завжди вказувати властивість *transition-duration*. Якщо цього не зробити, тривалість переходу буде *0s*, тому перехід не матиме жодного ефекту.

3. Визначити **ініціатор дії і фінальне значення** анімованої властивості *background-color*. Ініціатор – це дія, що призводить

до зміни одного стилю на інший. У CSS для запуску переходу використовують псевдокласи.

У наведеному прикладі для створення переходу використовується стиль для псевдокласу *:hover*, тому при наведенні покажчика миші на елемент *div*, буде спрацьовувати перехід.

Зауважимо, що анімувати можна багато різних властивостей, напр. *color, background-color, border-color*. Повний список властивостей CSS, які піддаються анімації, можна знайти за посиланням www.w3.org/TR/css3-transitions/#animatable-properties.

Анімація набору властивостей. За необхідності можна анімувати одночасно кілька властивостей CSS. У попередньому прикладі змінимо значення властивості *transition-property*:

```
transition-property:
        background-color, width, height, border-color;
```

та доповнимо кінцевий стиль анімації наступним чином:

```
div:hover { background-color: red;
            width: 120px;  height: 120px;  border-color: blue; }
```

Анімуються одночасно чотири властивості, анімація для них триває 2 секунди, але можна для кожної властивості вказати свою тривалість:

```
transition-property:
        background-color, width, height, border-color;
transition-duration: 2s, 3s, 1s, 2s;
```

Аналогічно до того, як у властивості *transition-property* через кому перераховуються анімовані властивості, у **властивості *transition-duration*** перераховуються через кому часові періоди анімації цих властивостей. Причому зіставлення часу певній властивості виконується згідно з позицією, тобто властивість *width* буде анімуватися 3 секунди.

Зауважимо, що замість перерахування через кому всіх анімованих властивостей як значення властивості *transition-property* можна вказати **ключове слово *all***. У такому разі будуть анімовані всі властивості, які змінюють значення в стилі для псевдокласу *:hover*:

```
transition-property: all;
transition-duration: 2s;
```

Властивість *transition-timing-function* дозволяє задати швидкість переходу і виконання анімації. Ця властивість відпо-

відає за те, як і в які моменти часу анімація буде прискорюватися або сповільнюватися.

Як значення властивість може приймати одну з функцій:

• *linear*: лінійна функція плавності: зміна властивості відбувається рівномірно по часу;

• *ease*: функція плавності, при якій анімація прискорюється до середини і сповільнюється до кінця, виглядаючи більш природнім чином;

• *ease-in*: функція плавності, при якій відбувається прискорення тільки на початку;

• *ease-out*: функція плавності, при якій відбувається прискорення тільки в кінці анімації;

• *ease-in-out*: функція плавності, при якій анімація починається і закінчується повільно, дещо прискорюючись всередині;

• *cubic-bezier*: для анімації застосовується кубічна функція Без'є. Для використання кубічної кривої Без'є у функцію *cubic-bezier* необхідно передати набір значень, напр..:

transition-timing-function: cubic-bezier(.5, .6, .24, .18);

Доповнимо стиль елемента *div* властивістю *transition-timing-function*:

```
div {   width: 100px; height: 100px;
        margin: 30px 30px;
        background-color: grey;
        transition-property: background-color;
        transition-duration: 2s;
        transition-timing-function: ease-in-out;
    }
div:hover{ background-color: red; }
```

Властивість *transition-delay* дозволяє визначити затримку перед виконанням переходу, часовий період вказується в секундах (s) або мілісекундах (ms):

transition-delay: 500ms;

Властивість *transition* являє собою скорочений запис вищерозглянутих властивостей. Наприклад, наступний опис властивостей:

transition-property: background-color;
transition-duration: 3s;

transition-timing-function: ease-in-out;
transition-delay: 500ms;
рівносильний такому запису:
transition: background-color 3s ease-in-out 500ms;
При цьому важливий порядок запису значень.

3.4.3. Анімація

1. Визначення анімації

Анімація надає великі можливості зміни стилю елемента. При переході є **набір властивостей** з початковими значеннями, які має елемент до початку переходу, і кінцевими значеннями, які встановлюються під час переходу. Але при анімації можна визначити не тільки два набори значень – початковий і кінцевий, але і множину проміжних наборів значень.

Анімація ґрунтується на послідовній зміні **ключових кадрів** (keyframes). Кожен ключовий кадр визначає один набір значень для анімованих властивостей. І послідовна зміна таких ключових кадрів фактично є **анімацією**.

По суті, переходи застосовують ту саму модель: неявно визначаються два ключові кадри – початковий і кінцевий, а сам перехід являє собою перехід від початкового до кінцевого ключового кадру. Єдина відмінність анімації, у даному випадку, полягає в тому, що для неї можна визначити **множину проміжних ключових кадрів**.

У загальному випадку, **оголошення анімації** в CSS3 має такий вигляд:

```
@keyframes назва_анімації {
   from {  /* початкові значення властивостей CSS */  }
   to {  /* кінцеві значення властивостей CSS */  }
}
```

Після ключового слова ***@keyframes*** вказується назва анімації. Анімація може мати довільну назву. Далі, у фігурних дужках, визначаються як мінімум два ключові кадри. Після ключового слова ***from*** блок оголошення початкового ключового кадру, а після ***to*** в блоці визначається кінцевий ключовий кадр. Кожний ключовий кадр містить одну або декілька властивостей CSS, аналогічно тому, як створюється звичайний стиль.

Приклад. Визначимо анімацію для фонового кольору елемента *div*:

```
<style>
 @keyframes backgroundColorAnimation {
        from { background-color: red; }
        to { background-color: blue; }
  }
 div {  width: 100px;  height: 100px;
        margin: 30px 30px;
        background-color: grey;
        animation-name: backgroundColorAnimation;
        animation-duration: 2s;
  }
</style>
```

що буде застосовано до кожного елемента *div* на вебсторінці:

```
<body> … <div></div>…
</body>
```

Анімація *backgroundColorAnimation* забезпечує перехід від червоного кольору фону до синього при завантаженні вебсторінки. Після завершення анімації буде той колір, який визначений у стилі елемента *div*, тобто сірий.

Щоб прикріпити анімацію до елемента, у стилі елемента вказується **властивість *animation-name***, значення цієї властивості – назва застосовуваної анімації.

Властивость *animation-duration* дозволяє задати тривалість анімації в секундах або мілісекундах. У прикладі вище тривалість анімації – 2 секунди.

При такому визначенні анімація буде запускатися одразу після завантаження сторінки. Однак можна також **запускати анімацію в залежності від дій користувача**. Наприклад, використовуючи визначення стилю з псевдокласом *:hover*, можна задати запуск анімації при наведенні покажчика миші на елемент:

```
<style>
  @keyframes backgroundColorAnimation {
     from { background-color: red; }
     to { background-color: blue; }
  }
```

```
  div { width: 100px;  height: 100px;
        margin: 30px 30px;
        background-color:grey;
   }
  div:hover { animation-name: backgroundColorAnimation;
              animation-duration: 2s;
  }
</style>
```

2. Множина ключових кадрів

Анімація, крім двох стандартних ключових кадрів, дозволяє задіяти багато проміжних. Для **визначення проміжного кадру** вказується процентне значення періоду анімації, в якому цей кадр повинен застосовуватися:

```
@keyframes backgroundColorAnimation {
    from { background-color: red; }
    25% { background-color: yellow; }
    50% { background-color: green; }
    75% { background-color: blue; }
    to { background-color: violet; }
}
```

У прикладі вище анімація починається з червоного кольору. Через 25% тривалості анімації, колір зміниться на жовтий, ще через 25% – на зелений і так далі.

Також можна в одному ключовому кадрі **анімувати одночасно кілька властивостей**:

```
@keyframes backgroundColorAnimation {
   from { background-color: red; opacity: 0.2; }
   to { background-color: blue; opacity: 0.9; }
}
```

Можна визначити **кілька окремих анімацій і застосувати їх разом**:

```
@keyframes backgroundColorAnimation {
  from { background-color: red; }
  to { background-color: blue; }
}
@keyframes opacityAnimation {
  from {  opacity: 0.2; }
  to { opacity: 0.9; }
```

```
}
div {  width: 120px;  height: 120px;
        margin: 30px 30px;
        background-color:grey;
        animation-name:
              backgroundColorAnimation, opacityAnimation;
        animation-duration: 2s, 3s;
}
```

Властивость *animation-name* приймає параметр – назви анімацій перераховані через кому, і аналогічно, властивість *animation-duration* – тривалості цих анімацій. Назва анімації та її тривалість зіставляються по позиції, напр. анімація *opacityAnimation* триватиме 3 секунди.

3. Завершення анімації

У загальному випадку після завершення часового інтервалу, зазначеного у властивості *animation-duration*, завершується і виконання анімації. Однак за допомогою додаткових властивостей можна перевизначити цю поведінку.

Властивість *animation-iteration-count* визначає, скільки разів буде повторюватися анімація. Наприклад,

```
animation-iteration-count: 3;
        /* три повтори анімації посліль*/
animation-iteration-count: infinite;
```

Значення *infinite* означає повторювати анімацію нескінченну кількість разів.

При повторі анімація буде починатися знову з початкового ключового кадру. Але за допомогою **властивості *animation-direction: alternate;*** можна задати протилежний напрям анімації при повторі, тобто вона починатиметься з кінцевого ключового кадру і виконуватиметься перехід до початкового кадру:

```
animation-name:
        backgroundColorAnimation, opacityAnimation;
animation-duration: 2s, 3s;
animation-iteration-count: 3;
animation-direction: alternate;
```

Після завершення анімації браузер встановить для анімованого елемента стиль, який був до виконання анімації. Але **властивість *animation-fill-mode*** зі значенням ***forwards*** дозволяє за-

дати остаточне значення анімованої властивості елемента те її значення, яке було в останньому кадрі:

animation-fill-mode: forwards;

4. Затримка анімації

За допомогою властивості *animation-delay* можна задати час затримки (очікування перед запуском) анімації:

animation-name: NameOfAnimation;
animation-duration: 3s;
animation-delay: 2s; / затримка в 2 секунди */*

5. Функція плавності анімації

Аналогічно, як і для переходів, для анімації існує **властивість *animation-timing-function***, що дозволяє контролювати швидкість виконання анімації. Ця властивість може набувати тих же значень, що й відповідна властивість *transition-timing-function* для переходів: *linear*, *ease*, *ease-in*, *ease-out*, *ease-in-out*, *cubic-bezier*. Наприклад:

animation-timing-function: ease-in-out;

6. Властивість *animation* є скороченим способом визначення вище розглянутих властивостей:

animation: <animation-name> <animation-duration>
[<animation-timing-function>] [<animation-iteration-count>]
[<animation-direction>] [<animation-delay>]
[<animation-fill-mode>];

Перші два параметри, які задають назву і тривалість анімації обов'язкові, решта значень необов'язкові.

Візьмемо наступний набір властивостей:

animation-name: NameOfAnimation;
animation-duration: 3s;
animation-timing-function: ease-in-out;
animation-iteration-count: 3;
animation-direction: alternate;
animation-delay: 2s;
animation-fill-mode: forwards;

Цей набір рівносильний такому визначенню анімації:

animation: NameOfAnimation 3s ease-in-out 3
alternate 2s forwards;

3.5. Адаптивний дизайн

3.5.1. Вступ в адаптивний дизайн

Сьогодні все більшого поширення набувають різні гаджети – смартфони, планшети, "розумні" годинники та інші пристрої, які дозволяють повноцінно користуватися інтернетом. Це створює нові можливості з розвитку веб-сайтів, просування інформаційних послуг і т.д., але разом з тим з'являються і нові проблеми.

Головна проблема полягає в тому, що стандартна веб-сторінка буде по-різному виглядати на різних пристроях із різним розширенням екрану. Первинним рішенням цієї проблеми було створення спеціальних версій для мобільних пристроїв. Але веб-сторінки необхідно підлаштовувати не тільки під невеликі екрани смартфонів або планшетів, але і під величезні екрани повноформатних широкоекранних телевізорів або гігантських планшетів типу Surface Hub, які також можуть мати доступ до інтернету.

І для вирішення проблеми сумісності веб-сторінок із найрізноманітнішими розширеннями пристроїв виникла **концепція адаптивного дизайну**. Її суть полягає в тому, щоб належним чином масштабувати елементи веб-сторінки в залежності від ширини екрану.

Хоча досі можна зустріти ситуацію, коли для сайту створюється окрема мобільна версія, часто з префіксом *m*, наприклад *m.facebook.com*, однак концепція адаптивного дизайну стає все більш поширеною й домінуючою.

Тестування адаптивного дизайну

При розробці адаптивних веб-сторінок можуть виникати труднощі тестування, оскільки, як правило, розробка виконується на звичайних персональних комп'ютерах. Але багато сучасних браузерів дозволяють емулювати запуск веб-сторінки на тому чи іншому пристрої з різною шириною екрану. Наприклад, в ***Google Chrome*** потрібно перейти в меню *Інші інструменти → Інструменти розробника*. Після відкриття панелі розробника на початку рядка меню самої панелі потрібно натиснути на іконку мобільного телефону, і після цього можна буде емулювати відображення сторінки на різних пристроях – від невеликих

телефонів до широкоформатних телевізорів. При бажанні можна вибрати мобільний пристрій, для якого відображатиметься сторінка (у верхньому рядку налаштувань, за замовчуванням, значення *Responsive*), або навіть створити емуляцію якогось нового пристрою, якого немає у вбудованому списку.

Ще одне рішення полягає у використанні емуляторів мобільних пристроїв. Невеликий список подібних емуляторів можна знайти за адресою http://www.mobilexweb.com/emulators.

3.5.2. Метатег Viewport

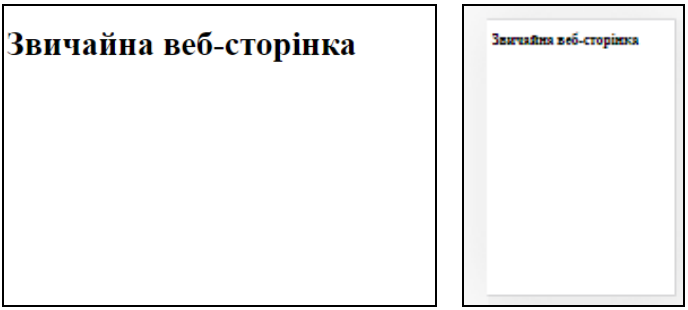
Перш за все розглянемо один із ключових моментів використання адаптивного дизайну – **метатег *viewport*** (фактично із цього тега починається адаптивний дизайн). Нехай є наступна веб-сторінка:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Звичайна веб-сторінка </title>
  </head>
<body>
   <h2> Звичайна веб-сторінка </h2>
</body>
</html>
```

Це стандартна веб-сторінка, яка у звичайному браузері буде виглядати звичним для нас чином (рис. 3.48а). Однак якщо запустити ту ж саму веб-сторінку в емуляторі мобільного пристрою або на реальному мобільному пристрої, то тільки докладаючи зусиль, можна прочитати, що на ній написано (рис. 3.48б).

Застосовуючи масштабування, користувач може нарешті побачити, що ж там все таки написано. Але це не зручно. При цьому веб-сторінка має багато порожнього місця, що не вельми красиво.

Чому так відбувається? Річ у тім, що кожен мобільний браузер задає сторінці деякі початкові розміри і потім намагається пристосувати сторінку під розміри екрану поточного мобільного пристрою.



а) *б)*

Рис. 3.48. Відображення звичайної веб-сторінки:
а – у браузері ПК; *б – на мобільному пристрої.*

Уся видима область на екрані мобільного браузера описується **поняттям *Viewport***. Фактично *viewport* являє собою область, в яку веб-браузер намагається "вписати" веб-сторінку. Наприклад, браузер Safari на iPone і iPod визначає ширину *viewport*, за замовчуванням, **980 пікселів**. Таким чином, браузер, отримавши сторінку і вписавши її у *viewport* із шириною 980 пікселів, стискає її до розмірів мобільного пристрою. Наприклад, якщо ширина екрану смартфона становить 320 пікселів, то до цього розміру буде стиснута сторінка і до всіх елементів сторінки буде застосовано коефіцієнт масштабування, що дорівнює 320/980.

Чому в даному випадку використовується саме 980 пікселів, а не реальний розмір екрану? Річ у тому, що, за замовчуванням, браузер вважає, що кожна веб-сторінка призначена для десктопів, а звичайною шириною десктопного сайту можна вважати 980 пікселів.

При цьому для кожного браузера встановлюється своя ширина області *viewport*, необов'язково 980 пікселів. Інші браузери можуть використовувати інші значення початкової ширини, але вони також будуть виконувати масштабування.

Щоб уникнути такої не вельми приємної картини, використовують **метатег *viewport***, який має наступне визначення:

<meta name="viewport" content="параметри_метатегу">

В **атрибуті *content*** мета-тегу можна визначити такі параметри:

№	Параметр	Значення	Опис
1	*width*	цілочислове значення в пікселях або значення ***device-width***	задає **ширину** області *viewport*;
2	*height*	цілочислове значення в пікселях або значення ***device-height***	задає **висоту** області *viewport*;
3	*initial-scale*	Число з плаваючою точкою від 0.1 і вище	задає **коефіцієнт масштабування** початкового розміру *viewport*, значення 1.0 означає відсутність масштабування;
4	*user-scalable*	no/yes	вказує, чи може користувач за допомогою жестів масштабувати сторінку;
5	*minimum-scale*	Число з плаваючою точкою від 0.1 і вище	задає мінімальний масштаб розміру *viewport*, значення 1.0 означає відсутність масштабування;
6	*maximum-scale*	Число з плаваючою точкою від 0.1 і вище	задає максимальний масштаб розміру *viewport*, значення 1.0 задає відсутність масштабування.

Доповнимо приклад веб-сторінки метатегом у розділі *head*:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Звичайна веб-сторінка</title>
</head>
<body>
   <h2> Звичайна веб-сторінка</h2>
</body>
</html>
```

Тепер при перегляді з мобільного пристрою сторінка матиме такий вигляд:

Очевидно, що веб-сторінка виглядає краще завдяки використанню метатега *viewport*.

Для веб-браузера пристрою **параметр *width=device-width*** встановлює, що значення початкової ширини області *viewport* потрібно взяти не 980 пікселів або якесь інше число, а безпосередню ширину екрану пристрою. Тому веб-браузер не проводитиме жодного масштабування, оскільки ширина *viewport* і ширина екрану будуть однакові.

Рис. 3.49. Веб-сторінка з метатегом *viewport*

Також можна використовувати інші параметри, наприклад, заборонити користувачеві збільшувати та зменшувати розміри сторінки:

```
<meta name="viewport" content="width=device-width,
      maximum-scale=1.0, minimum-scale=1.0">
```

3.5.3. Media Query в CSS

Іншим важливим елементом при побудові адаптивного дизайну є **правила Media Query**, які дозволяють визначити стиль в залежності від розмірів браузера користувача.

В **CSS2** було рішення у вигляді **правила media**, яке дозволяє вказати пристрій, для якого використовується певний стиль:

```
<html>
<head>
  <title>Адаптивна веб-сторінка</title>
  <link media="handheld" rel="stylesheet" href="mobile.css">
  <link media="screen" rel="stylesheet" href="desktop.css">
</head>
…
```

Атрибут ***rel="stylesheet"*** означає, що файл, який підключається, зберігає таблицю стилів (css).

Правило ***media="handheld"*** вказує, що стилі у *mobile.css* будуть застосовуватися для мобільних пристроїв, тоді як прави-

ло ***media="screen"*** використовуватиметься для десктопних браузерів.

Однак багато сучасних мобільних браузерів за замовчуванням вважають, що сторінка призначена для десктопів, і в будь-якому випадку застосовують правило *media="screen"*. Тому таке рішення ненадійне.

Для вирішення цієї проблеми в CSS3 введений **механізм CSS3 Media Query**. Наприклад, щоб застосувати стиль тільки для мобільних пристроїв, потрібно написати:

```
<head>
  <title>Адаптивна веб-сторінка</title>
  <meta name="viewport" content="width=device-width">
  <link rel="stylesheet" type="text/css" href="desctop.css" />
  <link rel="stylesheet" type="text/css"
      media="(max-device-width:480px)" href="mobile.css" />
 </head>
```

Значення **атрибуту media** *"max-device-width: 480px"* означає, що стилі з файлу *mobile.css* будуть застосовуватися до тих пристроїв, максимальна ширина екрану яких становить 480 пікселів. Фактично це і є мобільні пристрої.

За допомогою **ключового слова *and*** можна комбінувати умови:

```
<link rel="stylesheet" type="text/css"
      media="(min-width:481px) and (max-width:768px)"
      href="mobile.css" />
/* стиль буде застосовано, якщо ширина браузера знаходиться в діапазоні від 481 до 768 пікселів */
```

За допомогою **директиви *@import*** можна визначити один css-файл і імпортувати в нього стилі для певних пристроїв:

```
@import url(desctop.css);
@import url(tablet.css) (min-device-width:481px)
            and (max-device-width:768);
@import url(mobile.css) (max-device-width:480px);
```

Також можна не розділяти стилі по файлах, а використовувати **правила CSS3 Media Query в одному файлі css**:

```
body { background-color: red; }
    /*Далі решта стилів*/
@media (max-device-width:480px)
```

```
{   body { background-color: blue; }
    /*Далі решта стилів*/
}
```

Використовувані функції в CSS3 Media Query:

• *aspect-ratio*: відношення ширини до висоти області відображення (браузера);

• *device-aspect-ratio*: відношення ширини до висоти екрану пристрою;

• *max-width / min-width і max-height / min-height*: максимальна і мінімальна ширина та висота області відображення (браузера);

• *max-device-width / min-device-width і max-device-height / min-device-height*: максимальна і мінімальна ширина та висота екрану мобільного пристрою;

• *orientation*: орієнтація (портретна або альбомна).

Наприклад, можна задати різні стилі для різних орієнтацій мобільних пристроїв:

```
@media only screen and (orientation: portrait){
    /*Стилі для портретної орієнтації*/ }
@media only screen and (orientation: landscape){
   /*Стилі альбомної орієнтації */ }
```

Отже, ми змінюємо тільки визначення стилів у залежності від пристрою, а самі стилі css, по суті, залишаються тими ж, що й для створення звичайних сайтів.

Як правило, при визначенні стилів перевага надається стилям для найменших екранів – так званий **підхід Mobile First**, хоча це необов'язково. Наприклад, визначимо наступну вебсторінку:

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Адаптивна веб-сторінка</title>
  <style>
  body { background-color: red;
            /* для мобільних пристроїв */}
```

```
@media (min-width: 481px) and (max-width:768px) {
        /* для планшетів і фаблетів */
        body { background-color: green; } }
@media (min-width:769px) {
        /* для декстопів */
        body { background-color: blue;  }  }
</style>
</head>
<body>
  <h2>Адаптивна веб-сторінка</h2>
</body>
</html>
```

Визначена сторінка матиме червоний колір у випадку мобільного пристрою, зелений – для планшетів та фаблетів, у браузері звичайного комп'ютера сторінка матиме синій колір.

Спочатку розміщуються загальні стилі, які актуальні насамперед, для **мобільних пристроїв із невеликими екранами**. Потім стилі для пристроїв з екранами **середньої ширини**: фаблет, планшети, і далі стилі для **десктопів**.

3.6. Мультимедіа

3.6.1. Відео

Для відтворення відео в HTML5 використовується **елемент *video***, який має наступні атрибути:

- *src*: джерело відео – будь-який відеофайл;
- *width*: ширина елемента;
- *height*: висота елемента;
- *controls*: додає елементи управління відтворенням;
- *autoplay*: встановлює автовідтворення;
- *loop*: задає повторення відео;
- *muted*: відключає звук за замовчуванням;
- *preload*: управляє процесом завантаження відео;
- *poster*: визначає зображення, що буде відображатися до запуску відео.

Хоча можна задати ширину і висоту елемента, але вони не матимуть жодного впливу на аспектне відношення ширини і ви-

соти самого відео. Наприклад, якщо відео має формат 375×240, то при налаштуваннях *width="375" height="280"* воно буде центруватися на 280-піксельному просторі в HTML. Це дозволяє уникнути спотворень, які можуть виникнути при розтягуванні відео.

Наприклад:

<video src="my.mp4" width="400" height="300" controls >
</video>

При використанні **атрибутів *autoplay* і *loop*** одночасно, відео буде відтворюватися нескінченну кількість разів:

<video src="my.mp4" width="400" height="300"
controls autoplay loop ></video>

Щоб відключити звук, за замовчуванням, використовується **атрибут *muted***:

<video src="my.mp4" width="400" height="300"
controls muted >
</video>

Атрибут *preload* призначений управляти завантаженням відео; може набувати таких значень:

- *auto*: відео і пов'язані з ним метадані будуть завантажуватися до того, як відео почне відтворюватися;
- *none*: відео не буде завантажуватися у фоновому режимі, поки користувач не натисне на кнопку початку відтворення;
- *metadata*: у фоновому режимі до відтворення будуть завантажуватися тільки метадані (дані про формат, тривалость і т.д), саме відео не завантажуватиметься.

Наприклад:

<video src="my.mp4" width="400" height="300" controls
preload="auto"></video>

Атрибут *poster* дозволяє задати зображення, яке буде відображатися до запуску відео. Цьому атрибуту як значення передається шлях до зображення:

<video src="my.mp4" width="400" height="300" controls
poster="my.jpg"> </video>

Підтримка форматів відео. Головною проблемою при використанні елемента *video* є підтримка різними веб-браузерами певних форматів. За допомогою **вкладених**

елементів *source* можна задати кілька джерел відео, одне з яких буде використовуватися:

```
<video width="400" height="300" controls>
  <source src="my.mp4" type="video/mp4">
  <source src="my.webm" type="video/webm">
  <source src="my.ogv" type="video/ogg">
</video>
```

Елемент *source* має два атрибути для задання джерела відео:

- *src*: шлях до відеофайлу;
- *type*: тип відео (MIME-тип).

Якщо браузер не підтримує перший тип відео, то намагатиметься завантажити другий відеофайл. Якщо ж і тип другого відеофайлу не підтримується, то браузер звертається до третього відеофайлу.

3.6.2. Аудіо

Для відтворення звуку без відео в HTML5 використовується **елемент *audio***, який багато в чому подібний до елемента *video*. Для налаштування елемента *audio* використовуються **його атрибути**:

- *src*: шлях до аудіофайлу;
- *controls*: додає елементи управління відтворенням;
- *autoplay*: встановлює автовідтворення;
- *loop*: задає повторення аудіофайлу;
- *muted*: відключає звук за замовчуванням;
- *preload*: встановлює режим завантаження файлу.

Дія всіх цих атрибутів аналогічна їх дії в елементі *video*.

Наприклад:

```
<audio src="my.mp3" controls></audio>
```

В залежності від браузера, зовнішній вигляд елементів управління може відрізнятися. Ключовим моментом роботи з аудіо є підтримка браузером форматів файлів. На даний момент переважна більшість браузерів підтримують *mp3*. Однак якщо є сумніви, що аудіо в певному форматі буде підтримуватися браузером користувача, то можна використовувати вкладений **елемент *source*** і вказати аудіо в інших форматах:

```
<audio width="400" height="200" controls>
   <source src="my.mp3" type="audio/mpeg">
   <source src="my.m4a" type="audio/aac">
   <source src="my.ogg" type="audio/ogg">
</audio>
```

Як і у випадку з елементом *video*, у елемента *source* встановлюється атрибут *src* з посиланням на файл і атрибут *type* - тип файлу.

3.7. Flexbox

3.7.1. Що таке Flexbox. Flex Container

Flexbox – це загальна назва для модуля **Flexible Box Layout**, який є в CSS3. Цей модуль визначає особливий режим компоновки/верстки користувацького інтерфейсу, який називається **flex layout**. У цьому плані Flexbox надає інший підхід до створення користувацького інтерфейсу, який відрізняється від табличної або блокової верстки. Розгорнутий опис стандарту модуля можна подивитися в специфікації за посиланням https://drafts.csswg.org/css-flexbox/.

Завдяки Flexbox стає простіше створювати складні, комплексні інтерфейси, де можна легко перевизначити напрямок і вирівнювання елементів, створювати адаптивні табличні представлення. Крім того, Flexbox досить простий у використанні.

Основними складовими компонування flexbox є flex-контейнер (flex container) і flex-елементи (flex items). **Flex-контейнер** – це певний елемент, всередині якого розміщені **flex-елементи**.

Розглянемо основні поняття flexbox-верстки.

Одним із ключових понять є ***main axis***, або **центральна вісь.** Це умовна вісь у flex-контейнері, вздовж якої позиціонуються flex-елементи.

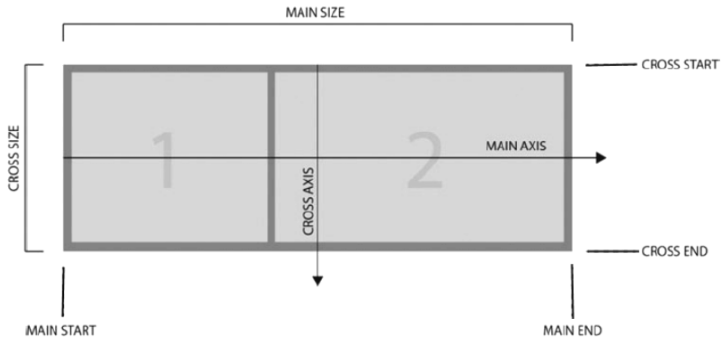


Рис. 3.50. Центральна та поперечна вісь у flexbox – верстці

Елементи у контейнері можуть розташовуватися **по горизонталі** у вигляді рядка або **по вертикалі** у вигляді стовпця. Залежно від типу розташування буде змінюватися і **центральна вісь**. Якщо розташування у вигляді рядка, то центральна вісь спрямована горизонтально зліва направо, якщо у вигляді стовпця – центральна вісь спрямована вертикально зверху вниз.

Терміни ***main start*** і ***main end*** описують, відповідно, початок і кінець центральної осі, а відстань між ними позначається як ***main size***.

Крім основної осі, існує також **поперечна вісь**, або ***cross axis***, вона перпендикулярна основній. При розташуванні елементів у вигляді рядка *cross axis* спрямована зверху вниз, а при розташуванні у вигляді стовпця – зліва направо. Початок поперечної осі позначається як ***cross start***, а її кінець – як ***cross end***. Відстань між ними описується терміном ***cross size***.

Тобто, якщо елементи розташовуються **в рядок**, то *main size* визначатиме ширину контейнера або елементів, а *cross size* – їх висоту. Якщо ж елементи розташовуються **в стовпець**, то, навпаки, *main size* визначає висоту контейнера і елементів, а *cross size* – їх ширину.

Для створення flex-контейнера потрібно присвоїти його стильовій **властивості *display*** одне з двох значень: *flex* або *inline-flex*. Значення ***flex*** визначає контейнер як блоковий елемент, а значення ***inline-flex*** визначає елемент як рядковий (*inline*). Розглянемо обидва способи на прикладі:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3</title>
  <style>
   .flex-container {  display: flex;
                      border:3px solid black; }
   .inline-flex-container {  display: inline-flex;
                     border:3px solid black; }
   .flex-item {  text-align:center;  font-size: 110%;
                color: white;  padding: 1.0em;  }
   .color1 {background-color: #70AEED;}
   .color2 {background-color: #D5D5E3;}
   .color3 {background-color: #E14068;}
  </style>
 </head>
 <body>
  <h3>display: flex</h3>
  <div class="flex-container">
   <!-- блоковий flex-контейнер -->
   <div class="flex-item color1">Flex Item 1</div>
   <div class="flex-item color2">Flex Item 2</div>
   <div class="flex-item color3">Flex Item 3</div>
  </div>
  <h3>display: inline-flex</h3>
  <div class="inline-flex-container">
   <!-- рядковий flex-контейнер -->
   <div class="flex-item color1">Flex Item 1</div>
   <div class="flex-item color2">Flex Item 2</div>
   <div class="flex-item color3">Flex Item 3</div>
  </div>
 </body>
</html>
```

У першому випадку flex-контейнер розтягується по ширині сторінки, а в другому – займає саме стільки місця, скільки необхідно для flex-елементів:

Рис. 3.51. Flex- контейнер

3.7.2. Напрямок flex-direction

Flex-елементи у flex-контейнері мають певний напрям, а саме можуть розташовуватися у вигляді рядків або у вигляді стовпців. Для задання напряму елементів CSS3 надає **властивість *flex-direction***, яка визначає напрям елементів і може набувати таких значень:

- ***row***: значення, за замовчуванням, при якому елементи розташовуються у вигляді рядка зліва направо;
- ***row-reverse***: елементи розташовуються у вигляді рядка тільки у зворотному порядку справа наліво;
- ***column***: елементи розташовуються в стовпець зверху вниз;
- ***column-reverse***: елементи розташовуються в стовпець у зворотному порядку від низу до верху.

Приклад розташування у вигляді рядка:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3</title>
  <style>
   .flex-container { display: flex;
                     border:3px solid grey; }
   .row { flex-direction: row; }
```

```
  .row-reverse { flex-direction: row-reverse; }
  .flex-item { text-align:center; font-size: 110%;
               color: white; padding: 1em; }
  .color1 { background-color: #70AEED;}
  .color2 { background-color: #D5D5E3;}
  .color3 {  background-color: #E14068;}
 </style>
</head>
<body>
  <h3> flex-direction: row</h3>
  <div class="flex-container row">
    <div class="flex-item color1">Flex Item 1</div>
    <div class="flex-item color2">Flex Item 2</div>
    <div class="flex-item color3">Flex Item 3</div>
  </div>
  <h3> flex-direction: row-reverse</h3>
  <div class="flex-container row-reverse">
    <div class="flex-item color1">Flex Item 1</div>
    <div class="flex-item color2">Flex Item 2</div>
    <div class="flex-item color3">Flex Item 3</div>
  </div>
</body>
</html>
```

Рис. 3.52. Розташування flex-елементів у вигляді рядка

Аналогічно розташування **у вигляді стовпця**:

*.column { **flex-direction: column;** }*

*.column-reverse { **flex-direction: column-reverse;** }*

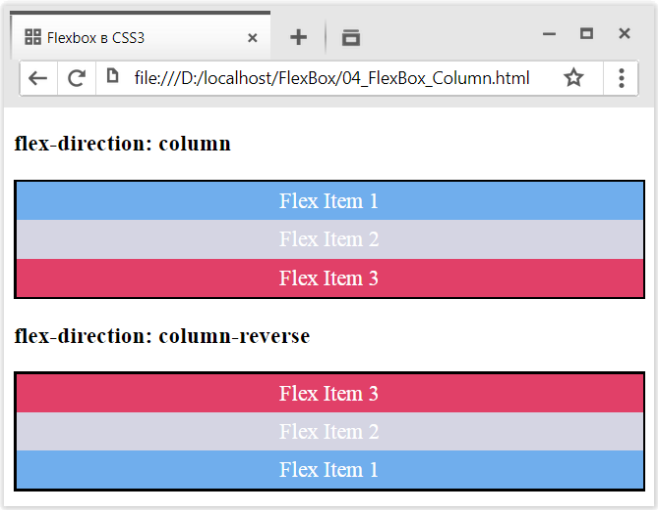


Рис. 3.53. Розташування flex-елементів у вигляді стовпця

3.7.3. Властивість flex-wrap

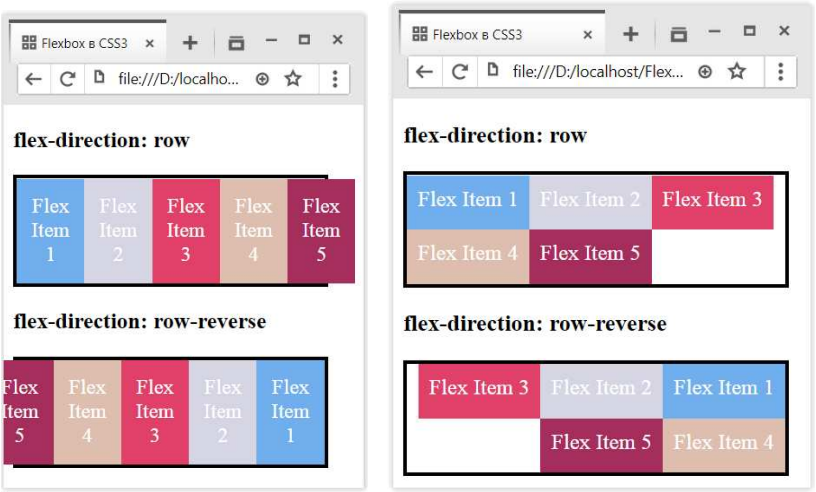
Властивість *flex-wrap* визначає, чи буде *flex*-контейнер містити кілька рядів (стовпців) елементів у випадку, якщо його розміри недостатні, щоб вмістити в один ряд (стовпець) усі елементи. Ця властивість може набувати таких значень:

- ***nowrap***: значення за замовчуванням, визначає *flex*-контейнер, де всі елементи розташовуються в один рядок (при розташуванні у вигляді рядків) або в один стовпець (при розташуванні в стовпець);
- ***wrap***: якщо елементи не вміщуються у *flex*-контейнер, то створює додаткові ряди (стовпці) в контейнері для розміщення елементів: при розташуванні у вигляді рядка створюються додаткові рядки, а при розташуванні у вигляді стовпця – додаткові стовпці;
- ***wrap-reverse***: те саме, що й значення *wrap*, тільки елементи розташовуються у зворотному порядку.

Доповнимо стиль *flex*-контейнера з попереднього прикладу властивістю *flex-wrap*:

```
.flex-container { display: flex; flex-wrap:nowrap;
                  width: 90%;  height:5.5em;
                  border: 3px solid black; }
.row { flex-direction: row; }
.row-reverse { flex-direction: row-reverse; }
.flex-item {   text-align:center; font-size: 110%;
               color: white; padding: 0.5em; }
```

а сам *flex*-контейнер у тілі веб-сторінки – ще декількома елементами для ілюстрації значень властивості ***flex-wrap***:



а) *б)*

Рис. 3.54. Властивість ***flex-wrap:***
а – значення ***nowrap;*** *б* – значення ***wrap.***

У випадку значення *nowrap*, в кожному з *flex*-контейнерів по п’ять елементів. Однак ширина контейнера може не вміщати всі елементи, тоді вони виходять за межі контейнера.

При заданні значення *wrap*, *flex*-контейнер доповнюється додатковими рядами для розміщення всіх елементів у контейнері.

3.7.4. Властивість flex-flow. Порядок елементів

Властивість *flex-flow* дозволяє задати значення одночасно для властивостей *flex-direction* і *flex-wrap*; має синтаксис:

flex-flow: <flex-direction> [<flex-wrap>]

де друга властивість *flex-wrap* необов'язкова. Якщо її опустити, для неї буде використовуватися значення за замовчуванням – *nowrap*.

Відповідно, властивість *flex-flow* має значення за замовчуванням:

row nowrap;

Властивість order

У *flex*-контейнері можна змінити порядок розташування *flex*-елементів, використовуючи **властивіть *order***.

Властивість *order* дозволяє розподілити *flex*-елементи по групах. До однієї групи може належати кілька елементів. Кожна група позначається номером – ціле число – значення властивості *order*. За замовчуванням, якщо для *flex*-елемента явно не вказано властивість *order*, то її значення 0.

При розташуванні елементів у *flex*-контейнері спочатку впорядковуються групи елементів за зростанням їх номеру, а елементи, всередині групи, виводяться на веб-сторінку в порядку їх розташування в html-коді.

Наприклад, спочатку розташовуються елементи з групи 0, наступні – елементи з групи 1, а останні – елементи з групи 2 і так далі, якщо порядковий номер кожної групи є додатним числом.

Нехай визначено три групи *flex*-елементів з номерами: -1, 1 та елементи, для яких не вказана група (властивість *order*), тому вони формують групу під номером 0:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3</title>
  <style>
   .flex-container { display: flex;  flex-flow:row wrap;
                     border: 3px solid black; }
   .flex-item { /*стиль flex-елемента */
```

```
        text-align:center; font-size: 120%;
        padding: 0.8em; color: white; }
    .group1 { order:-1; }
    .group2 { order: 1; }
    .color1 { background-color: #70AEED;}
    .color2 { background-color: #D5D5E3;}
    .color3 { background-color: #E14068;}
    .color4 { background-color: #DEBFAF;}
    .color5 { background-color: #A62E5C;}
  </style>
 </head>
 <body>
  <h3>Властивість order: зміна порядку
      розташування елементів</h3>
  <div class="flex-container">
     <div class="flex-item color1">Flex Item 1</div>
     <div class="flex-item color2 group2">Flex Item 2</div>
     <div class="flex-item color3 group2">Flex Item 3</div>
     <div class="flex-item color4">Flex Item 4</div>
     <div class="flex-item color5 group1">Flex Item 5</div>
  </div>
 </body>
</html>
```

У цьому випадку, першим відображається елемент 5, оскільки він належить до групи -1, далі з групи 0: елементи 1 і 4, і остання група з номером 1: елементи: 2 і 3. Отже, отримуємо:

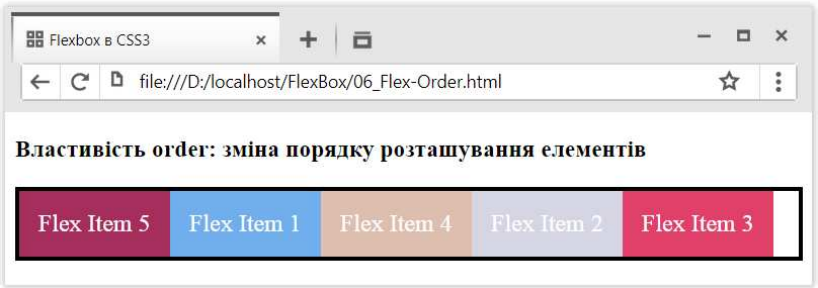


Рис. 3.55. Властивість order: впорядкування елементів

3.7.5. Вирівнювання елементів. Властивість justify-content

Можлива ситуація, коли простір *flex*-контейнера за розміром відрізняється від простору, необхідного для *flex*-елементів. Наприклад:

- *flex*-елементи не використовують весь простір *flex*-контейнера;
- *flex*-елементам потрібно більше простору, ніж є у *flex*-контейнері. В цьому випадку елементи виходять за межі контейнера.

Для управління цими ситуаціями можна використовувати ***властивість justify-content***, яка вирівнює елементи в контейнері вздовж основної осі *main axis* (при розташуванні у вигляді рядка по горизонталі, при розташуванні у вигляді стовпця – по вертикалі) і може набувати таких значень:

- ***flex-start***: значення за замовчуванням, елементи вирівнюються по лівому краю контейнера (при розташуванні у вигляді рядка) або по верху (при розташуванні у вигляді стовпця);
- ***flex-end***: елементи вирівнюється по правому краю (при розташуванні у вигляді рядка) або по низу (при розташуванні у вигляді стовпця) контейнера;
- ***center***: елементи вирівнюються по центру;
- ***space-between***: якщо в рядку тільки один елемент або елементи виходять за межі *flex*-контейнера, то дане значення аналогічно *flex-start*. В інших випадках, перший елемент вирівнюється по лівому краю (при розташуванні у вигляді рядка) або по верху (при розташуванні у вигляді стовпця), а останній елемент – по правому краю контейнера (при розташуванні у вигляді рядка) або по низу (при розташуванні у вигляді стовпця), решта простору рівномірно розподіляється між іншими елементами;
- ***space-around***: якщо в рядку тільки один елемент або елементи виходять за межі контейнера, то це значення аналогічно значенню *center*. В інших випадках, елементи рівномірно розподіляються в контейнері (однакова відстань між ними) так, щоб відстань від першого і останнього елементів до межі контейнера становила половину відстані між елементами;

• ***space-evenly***: якщо в рядку тільки один елемент або елементи виходять за межі контейнера, то це значення аналогічно значенню *center*. В інших випадках, елементи рівномірно розподіляються в контейнері так, щоб відстань між будь-якими двома сусідніми елементами і відстань від крайніх елементів до меж контейнера була однакова.

Наприклад, визначимо стиль flex-контейнера:

.flex-container { display: flex; justify-content: flex-end; }

Проілюструємо значення **властивості *justify-content*** для розташування елементів у вигляді рядків:

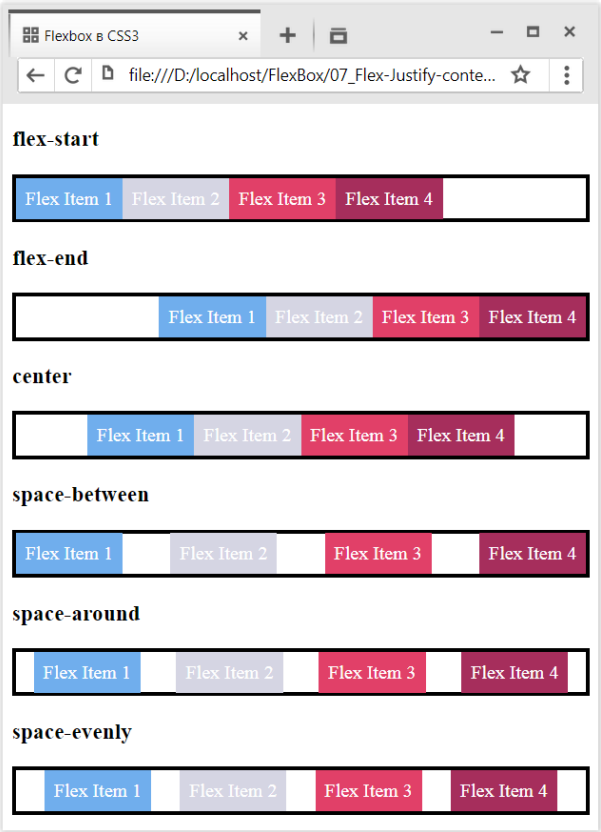


Рис. 3.56. Властивість *justify-content*: розташування рядками

Значення **властивості *justify-content*** для розташування елементів у вигляді стовпців:

Рис. 3.57. Властивість *justify-content*: розташування стовпцями

3.7.6. Вирівнювання елементів: align-items і align-self

Властивість *align-items* вирівнює елементи в контейнері, але по поперечній осі (cross axis): при розташуванні у вигляді рядка – по вертикалі, при розташуванні у вигляді стовпця – по горизонталі. Властивість може набувати таких значень:

- ***stretch***: значення за замовчуванням, *flex*-елементи розтягуються по всій висоті (при розташуванні в рядок) або по всій ширині (при розташуванні в стовпець) *flex*-контейнера;
- ***flex-start***: елементи вирівнюються по верхньому краю (при розташуванні в рядок) або по лівому краю (при розташуванні в стовпець) *flex*-контейнера;
- ***flex-end***: елементи вирівнюються по нижньому краю (при розташуванні в рядок) або по правому краю (при розташуванні в стовпець) *flex*-контейнера;
- ***center***: елементи вирівнюються по центру *flex*-контейнера;
- ***baseline***: елементи вирівнюються відповідно до своєї базової лінії.

Наприклад, доповнимо стиль *flex*-контейнера властивістю *align-items*:

.flex-container { display: flex; align-items: flex-end; }

Проілюструємо значення **властивості *align-items*** при розташуванні *flex*-елементів в рядок:

Рис. 3.58. Властивість *align-items:* розташування рядками

Аналогічно властивість працює при розташуванні елементів у стовпець:

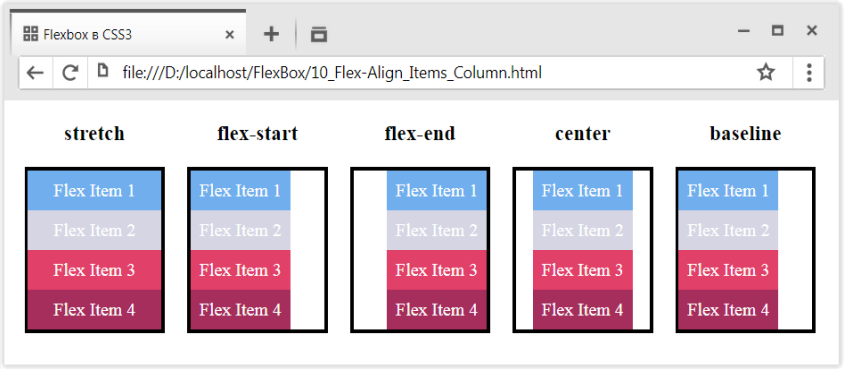


Рис. 3.59. Властивість *align-items:* розташування стовпцями

Властивість *align-self* дозволяє перевизначити значення властивості *align-items* для одного елемента. Властивість приймає **ті самі значення: *stretch, flex-start, flex-end, center, baseline*** та значення "***auto***": значення за замовчуванням, при якому елемент отримує значення від властивості *align-items*, яка визначена для *flex*-контейнера; якщо у контейнері такий стиль не визначений, то застосовується значення *stretch*.

Наприклад, для *flex*-контейнера задано розтягування по висоті за допомогою значення *align-items: stretch;*, але для кожного *flex*-елемента цю поведінку можна перевизначити:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3</title>
  <style>
   .flex-container { /* стиль flex- контейнера */
      display: flex;  justify-content: space-between;
      align-items: stretch;
      border:3px solid black;
      height:7.0em; }
   .flex-item { /* стиль flex- елемента*/ …
         padding: 0.7em;  text-align:center;
         font-size: 1.2em; color: white; }
   .item1 { align-self: stretch;
```

```
            background-color: #B8C6D1; }
    .item2 { align-self: center;
            background-color: #70AEED; }
    .item3 { align-self: flex-start;
            background-color: #70E393; }
    .item4 { align-self: flex-end;
            background-color: #E14068; }
    .item5 { align-self: baseline;
            background-color: #A62E5C; }
    .item6 { align-self: auto;
             background-color: #88A404; }
  </style>
 </head>
 <body>
   <div class="flex-container">
     <div class="flex-item item1">stretch</div>
     <div class="flex-item item2">center</div>
     <div class="flex-item item3">flex-start</div>
     <div class="flex-item item4">flex-end</div>
     <div class="flex-item item5">baseline</div>
     <div class="flex-item item6">auto</div>
   </div>
 </body>
</html>
```

Отримуємо ілюстрацію значення властивості *align-self* у випадку розташування *flex*-елементів у рядок:

Рис. 3.60. Властивість *align-self*

Отже, **властивість *align-self*** перевизначає вирівнювання по поперечній осі для окремих елементів у *flex*-контейнері.

3.7.7. Вирівнювання рядків і стовпців: align-content

Властивість *align-content* визначає вирівнювання рядів (стовпців), якщо розташування елементів у *flex*-контейнері в рядок (стовпець) і застосовується, якщо властивість *flex-wrap* має значення *wrap* або *wrap-reverse*. Властивість *align-content* може мати такі значення:

- ***stretch***: значення за замовчуванням, рядки (стовпці) розтягуються, займаючи весь вільний простір;
- ***flex-start***: рядки (стовпці) вирівнюються по початку контейнера (для рядків – це верхній край, для стовпців – лівий край контейнера);
- ***flex-end***: рядки (стовпці) вирівнюються по кінцю контейнера (рядки - по нижньому краю, стовпці - по правому краю);
- ***center***: рядки (стовпці) позиціонуються по центру контейнера;
- ***space-between***: рядки (стовпці) рівномірно розподіляються по контейнеру, з однаковими відступами між ними; відступи між першим і останнім рядками (стовпцями) та межею контейнера відсутні; якщо ж наявного в контейнері місця недостатньо, то діє аналогічно значенню *flex-start*;
- ***space-around***: рядки (стовпці) рівномірно розподіляються по контейнеру з однаковими відступами між ними так, щоб відстань між першим і останнім рядками (стовпцями) та межею контейнера становила половину відстані між двома сусідніми рядками (стовпцями).

Потрібно враховувати, що ця властивість має сенс, якщо в контейнері два і більше рядків (стовпців).

Наприклад, стиль *flex*-контейнера:

```
.flex-container {  display: flex;  flex-wrap: wrap;
                   align-content: space-between;
                   border:3px solid black;
                   height:130px; }
```

Проілюструємо значення **властивості *align-content***, у випадку розташування вмістимого *flex*–контейнера у вигляді рядків:

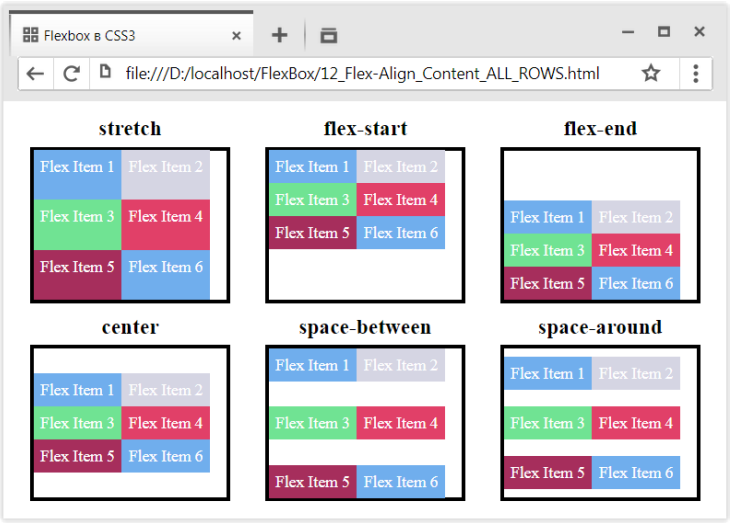


Рис. 3.61. Властивість *align-content:* розташування рядками

Якщо *flex*-контейнер доповнити властивістю *flex-direction: column;* отримаємо розташування *flex*-елементів у вигляді стовпців:

Рис. 3.62. Властивість *align-content:* розташування стовпцями

3.7.8. Управління елементами: flex-basis, flex-shrink і flex-grow

Крім властивостей, які визначають вирівнювання елементів відносно меж *flex*-контейнера, є ще три властивості, які дозволяють **управляти розміром *flex*-елементів**:

1. Властивість *flex-basis*

Flex-контейнер може збільшуватися або зменшуватися вздовж своєї центральної осі (*main axis*), наприклад, при зміні розмірів браузера, якщо контейнер має нефіксовані розміри. Разом із контейнером також можуть збільшуватися і зменшуватися його *flex*-елементи. **Властивість *flex-basis*** визначає початковий розмір *flex*-елемента до того, як він почне змінювати розмір, підлаштовуючись під розміри *flex*-контейнера.

Ця властивість може набувати таких значень:

- ***auto***: початковий розмір елемента встановлюється автоматично;
- ***content***: розмір елемента визначається за його вмістом, це значення підтримується не всіма сучасними браузерами, тому поки що варто його уникати;
- ***числове значення***: можна задати числове значення для розмірів елемента.

Розглянемо три різні випадки задання розмірів *flex*-елемента:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3</title>
  <style>
   h3 { text-align:right; margin-right:30px; }
   .flex-container { display: flex; width:600px;
                     border:3px solid black; }
   .flex-item { /*стиль flex- елемента*/
         text-align:center;  font-size: 1.1em;
        padding: 0.8em;  color: white;}
   /* стилі задання розмірів flex-елемента */
   .item1 { flex-basis: auto; width:150px;
```

```
            background-color: #70AEED; }
    .item2 { flex-basis: auto; width:auto;
            background-color: #D5D5E3; }
    .item3 { flex-basis: 200px; width:150px;
            background-color: #E14068;}
  </style>
 </head>
 <body>
  <h3>flex-basis визначає початковий розмір …</h3>
    <div class="flex-container">
        <div class="flex-item item1">Flex Item 1</div>
        <div class="flex-item item2">Flex Item 2</div>
        <div class="flex-item item3">Flex Item 3</div>
    </div>
 </body>
</html>
```

У першому випадку властивость *flex-basis* має значення ***auto***, тому для першого елемента реальним значенням ширини буде значення **властивості *width***.

Для другого елемента властивость *flex-basis* має значення ***auto***, однак і **властивість *width*** також має значення ***auto***, тому реальна ширина елемента буде встановлюватися **по його вмісту**.

У третього елемента властивість *flex-basis* має **числове значення**, яке і використовується, а властивість *width*, у цьому випадку, не має жодного значення.

Отже, *flex*-контейнер із визначеними в ньому елементами має вигляд:

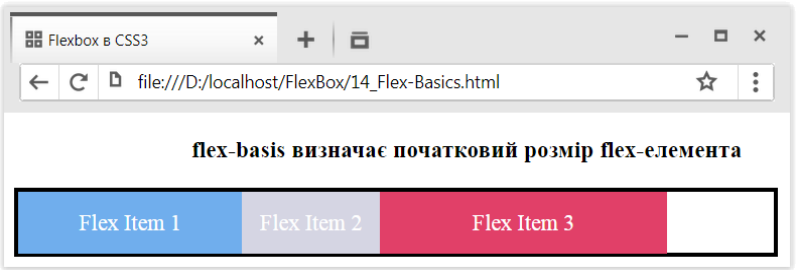


Рис. 3.63. Властивість *flex-basis*

2. Властивість *flex-shrink*

Якщо *flex*-контейнер має недостатньо місця для розміщення елементів, то буде виконуватися усікання (стиснення) елементів в контейнері згідно зі стандартними правилами масштабування.

При цьому, якщо потрібно перевизначити **співвідношення стиснення** елементів у контейнері по відношенню до інших елементів контейнера, застосовують властивість *flex-shrink*. **Властивість *flex-shrink*** дозволяє перевизначити, наскільки елемент буде усікатися (стискатися) відносно інших елементів.

Значенням властивості може бути будь-яке додатне число. За замовчуванням, значення 1. У такому випадку всі елементи контейнера будуть усікатися згідно з правилами масштабування.

Приклад. Нехай *flex*-контейнер має ширину **500px**, в ньому потрібно розмістити три елементи шириною **150**, **200**, **300px** відповідно. Для елементів задано властивість ***flex-shrink*** зі значеннями **1, 2 і 3** відповідно.

Сумарна ширина всіх елементів 150px+200px+300px=**650px** при наявній ширині контейнера в 500px. Отже, елементи, що містяться в контейнері, сумарно повинні бути усічені на **150px**. Ці 150px будуть компенсовані скороченням на певну кількість пікселів кожного з трьох елементів, причому кожен елемент скорочується відповідно до свого значення *flex-shrink*, помноженого на його ширину.

Спочатку обчислюється **загальна «вага»**:

1 × 150 + 2 × 200 + 3 × 300 = **1450**

Отже, **кожен елемент буде скорочений на:**

150 × 1(*flex-shrink*) × 150(*width*) / 1450 = 15.5;

150 × 2(*flex-shrink*) × 200(*width*) / 1450 = 41.4;

150 × 3(*flex-shrink*) × 300(*width*) / 1450 = 93.1;

де 150 – кількість пікселів, на які потрібно, сумарно, скоротити всі елементи.

І фінальні розміри елементів будуть такими:

150–15.5 = 134.5;

200–41.4 = 158.6;

300–93.1 = 206.9.

Отримуємо: 134.5+158.6+206.9=500.

Проілюструємо на прикладі:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3</title>
  <style>
   * { box-sizing: border-box; }
   h3 { text-align:center;
        margin-top:10px;
        margin-bottom:10px;  }
   .flex-container { display: flex; width:500px;
                     border:2px solid black; }
   .flex-item { /*стиль flex- елемента*/
         text-align:center;  font-size: 1.1em;
         padding: 0.7em;  color: white; }
     /*задання розмірів елементів та власт. flex-shrink */
   .item1 { flex-basis: 150px; flex-shrink:1;
            background-color: #70AEED;}
   .item2 { flex-basis: 200px; flex-shrink:2;
            background-color: #D5D5E3;}
  .item3 { flex-basis: 300px;  flex-shrink:3;
            background-color: #E14068;}
     /*задання тільки розмірів без власт. flex-shrink */
   .item10 { flex-basis: 150px; background-color: #70AEED; }
   .item20 { flex-basis: 200px; background-color: #D5D5E3; }
   .item30 { flex-basis: 300px; background-color: #E14068; }
  </style>
 </head>
 <body>
  <h3>flex-shrink не використовується</h3>
  <div class="flex-container">
        <div class="flex-item item10">Flex Item 1</div>
        <div class="flex-item item20">Flex Item 2</div>
        <div class="flex-item item30">Flex Item 3</div>
  </div>
  <h3>flex-shrink має значення: 1,2,3</h3>
  <div class="flex-container">
```

```
        <div class="flex-item item1">Flex Item 1</div>
        <div class="flex-item item2">Flex Item 2</div>
        <div class="flex-item item3">Flex Item 3</div>
    </div>
  </body>
</html>
```

Порівняємо отримані результати з використанням властивості *flex-shrink* та без неї (працює стандартне масштабування):

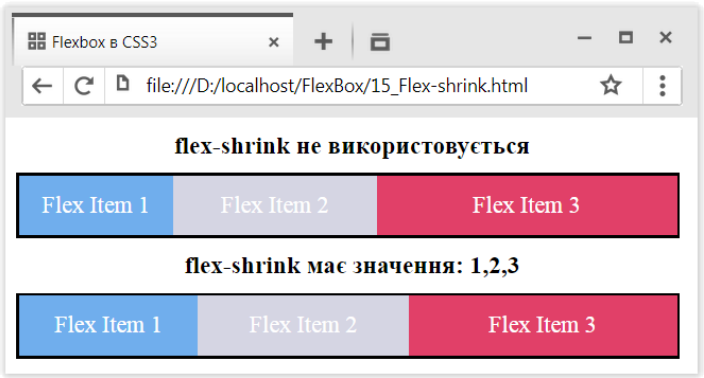


Рис. 3.64. Властивість *flex-shrink*

3. Властивість *flex-grow* визначає, як вільний простір *flex*-контейнера розподіляється серед його *flex*-елементів і наскільки велика частка кожного з них.

Іншими словами, *flex*-контейнер розподіляє вільне місце серед своїх елементів, пропорційно їх ***flex-grow* фактору**, або урізає їх, пропорційно їх ***flex-shrink*** фактору, для того того, щоб припинити переповнення контейнера.

Як значення властивість *flex-grow* приймає додатне число, яке вказує, у скільки разів елемент буде збільшуватися відносно інших елементів. **За замовчуванням, властивість *flex-grow* має значення 0**.

Проілюструємо використання властивості *flex-grow*:

```
<!DOCTYPE html>
<html>
 <head>
```

```
<meta charset="utf-8">
<title>Flexbox в CSS3</title>
<style>
 h3 { text-align:center; margin-top:10px;
      margin-bottom:10px; }
 .flex-container {  display: flex;
                    border:3px black solid; }
 .flex-item { /*стиль flex- елемента*/
      text-align:center;  font-size: 1.1em;
      padding: 0.6em;  color: white;}
 .item10 {background-color: #70AEED; }
 .item20 {background-color: #D5D5E3; }
 .item30 {background-color: #E14068; }
 .item1 { flex-grow:0; background-color: #70AEED; }
 .item2 { flex-grow:1; background-color: #D5D5E3; }
 .item3 { flex-grow:2; background-color: #E14068;  }
</style>
</head>
<body>
 <h3>flex-grow не використовується </h3>
 <div class="flex-container">
    <div class="flex-item item10">Flex Item 1</div>
    <div class="flex-item item20">Flex Item 2</div>
    <div class="flex-item item30">Flex Item 3</div>
 </div>
 <h3> flex-grow має значення: 0,1,2</h3>
 <div class="flex-container">
    <div class="flex-item item1">Flex Item 1</div>
    <div class="flex-item item2">Flex Item 2</div>
    <div class="flex-item item3">Flex Item 3</div>
 </div>
</body>
</html>
```

Отже, для кожного елемента є базові початкові розміри. У даному випадку, явно розміри для елементів не вказані, тому розмір кожного елемента буде складатися з розмірів внутрішнього вмісту, до яких додаються внутрішні відступи.

Елементи будуть збільшуватися, відповідно, до значення властивості *flex-grow*, яка визначена для кожного елемента. У першого елемента властивість *flex-grow* дорівнює 0, тому перший елемент має константні постійні розміри. У другого елемента *flex-grow* має значення 1, у третього – 2. Таким чином, в сумі 0+1+2=3. Тому другий елемент буде збільшуватися на 1/3 вільного простору, а третій елемент отримає 2/3 вільного простору.

Для наочності продемонструємо також той же контейнер із тими ж елементами, але без використання властивості *flex-grow:*

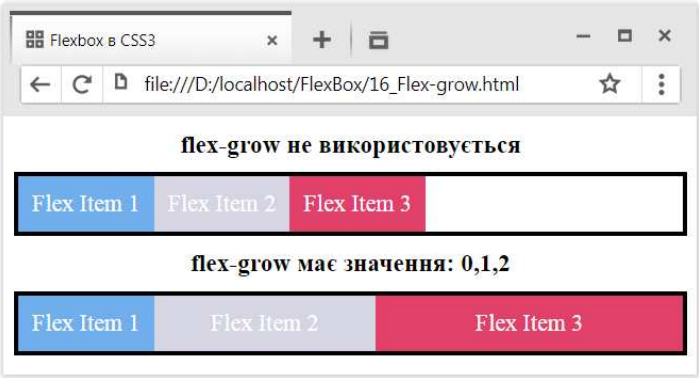


Рис. 3.65. Властивість *flex-grow*

4. Властивість *flex* є об'єднанням властивостей *flex-basis*, *flex-shrink* і *flex-grow* і має наступний синтаксис:

flex: <flex-grow> <flex-shrink> <flex-basis>;

За замовчуванням, властивість *flex* має значення "*0 1 auto*".

Крім конкретних значень для кожної з підвластивостей, **для властивості *flex* можна задати одне з трьох загальних значень:**

- ***none*** еквівалентно значенням "*0 0 auto*", при якому *flex*-елемент не розтягується і не буде скорочуватися при зменшенні контейнера;
- ***auto*** еквівалентно значенню "*1 1 auto*";
- ***initial*** еквівалентно значенню "*0 1 auto*".

Наприклад, застосуємо властивість *flex*:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3</title>
  <style>
   .flex-container { display: flex;
                     border:3px black solid; }
   .flex-item { /*стиль flex- елемента*/
        text-align:center;  font-size: 1.1em;
        padding: 0.6em;  color: white;}
    /*стилі flex-елементів */
   .item1 { flex: 0 0 auto; width: 150px;
            background-color: #70AEED;  }
   .item2 { flex: 1 0 auto; width: 150px;
            background-color: #D5D5E3;  }
   .item3 { flex: 0 1 auto; width: 150px;
            background-color: #E14068; }
   .item4 { flex: 1 1 auto; width: 150px;
            background-color: #59CA65; }
  </style>
 </head>
 <body>
  <div class="flex-container">
   <div class="flex-item item1">Flex Item 1</div>
   <div class="flex-item item2">Flex Item 2</div>
   <div class="flex-item item3">Flex Item 3</div>
   <div class="flex-item item4">Flex Item 4</div>
  </div>
 </body>
</html>
```

Для всіх 4-х елементів *flex*-контейнера задано значення підвластивості *flex-basis* як *auto*, тому їх початкові розміри будуть підібрані автоматично.

При розтягуванні вікна розміри першого і третього елементів не зміняться, а вільний простір буде розподілено між другим і четвертим елементами рівномірно:

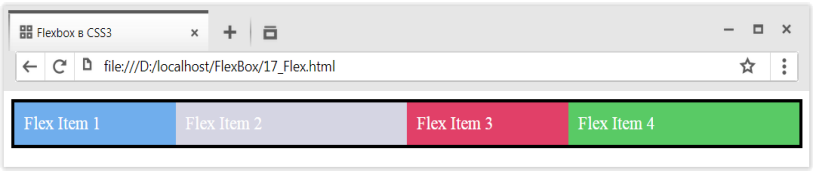


Рис. 3.66 а) Властивість *flex*: розтягування вікна браузера

При стисканні вікна розміри першого і другого елементів не зміняться, а третій і червертий – будуть стиснуті однаковою мірою:

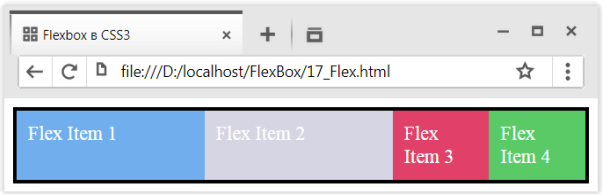


Рис. 3.66 б) Властивість *flex*: стискання вікна браузера

3.7.9. Багатоколонковий дизайн на Flexbox

Розглянемо, як можна зробити найпростіші багатоколонкові макети сторінки за допомогою Flexbox.

Макет сторінки з двоконковим дизайном:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3: Двоколонковий дизайн</title>
  <style>
     *{ box-sizing: border-box; }
     h2 {text-align:center;}
     html, body { padding: 0; margin: 0;
                  font-family: verdana, arial, sans-serif; }
     body { display: flex; flex-direction: column;
            padding: 1em; }
     .flex-item { background-color: #E4E4E6;
        font-size: 1.1em;  padding: 0.6em;  }
     .flex-item:nth-child(even) {
        background-color:#C1CDE5; }
```

```
    @media screen and (min-width: 600px) {
      body { flex-direction: row; min-height: 100vh; }
      }
 </style>
</head>
<body>
  <div class="flex-item">
       <h2>Що таке Lorem Ipsum?</h2>
       <p>Lorem Ipsum - це текст - "риба", часто … </p>
  </div>
  <div class="flex-item">
    <h2>Звідки він з`явився?</h2>
    <p> Багатохто думає, що Lorem Ipsum, взятий …</p>
   </div>
</body>
</html>
```

Сторінка має вигляд:

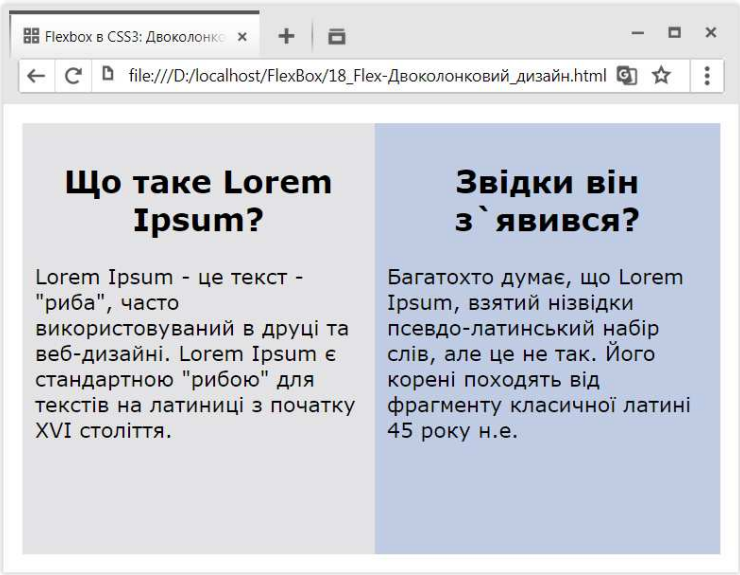


Рис. 3.67. Двоконковий дизайн

У прикладі **flex-контейнером** є елемент body. Оскільки на мобільних пристроях (особливо смартфонах) розмір екрану не такий великий, тому, за замовчуванням, встановлюємо розташування елементів у стовпець. Однак для пристроїв з екраном **від 600px і вище діє правило media-query**, яке встановлює розташування у вигляді рядка.

Розглянемо випадок **дизайну з трьох колонок**:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Flexbox в CSS3: Трьохколонковий режим</title>
  <style>
     *{ box-sizing: border-box; }
     h2 {text-align:center;}
     html, body { padding: 0; margin: 0;
                  font-family: verdana, arial, sans-serif; }
     body { display: flex; flex-direction: column;
            padding: 1em; }
     .flex-item { flex:1;
         background-color: #E4E4E6; /*grey*/
         font-size: 1.1em; padding: 0.6em;  }
    .flex-item:nth-child(1) {
        background-color:#C1CDE5; /*blue*/
        }
    @media screen and (min-width: 600px) {
       body { flex-direction: row; min-height: 100vh; }
       .flex-item:nth-child(2) { order:-1; }
      } /*end @media*/
  </style>
 </head>
 <body>
   <div class="flex-item">
     <h2>Що таке Lorem Ipsum?</h2>
     <p>Lorem Ipsum - це текст - "риба", часто …</p>
   </div>
   <div class="flex-item">
     <h2>Класичний текст Lorem Ipsum</h2>
```

```
    <p> "Lorem ipsum dolor sit amet, consectetur …</p>
  </div>
  <div class="flex-item">
    <h2>Звідки він з`явився?</h2>
    <p> Багатохто думає, що Lorem Ipsum, взятий …</p>
  </div>
 </body>
</html>
```

На відміну від попереднього прикладу, додано ще один елемент. Особливістю цього прикладу є те, що стовпці мають однакові розміри. Для цього у них встановлено властивість ***flex: 1***, тобто при розтягуванні або зменшенні границь контейнера всі елементи будуть масштабуватися на однакову величину.

Крім того, при **ширині екрану більше 600px** у другого елемента встановлюється властивість *order: -1*, завдяки чому цей елемент розміщується першим. Нагадаємо, що *:nth-child(2)* – псевдоклас, що забезпечує можливість стилізації дочірнього елемента, в даному випадку, другого, оскільки нумерація елементів від 1.

Веб-сторінка з трьома колонками:

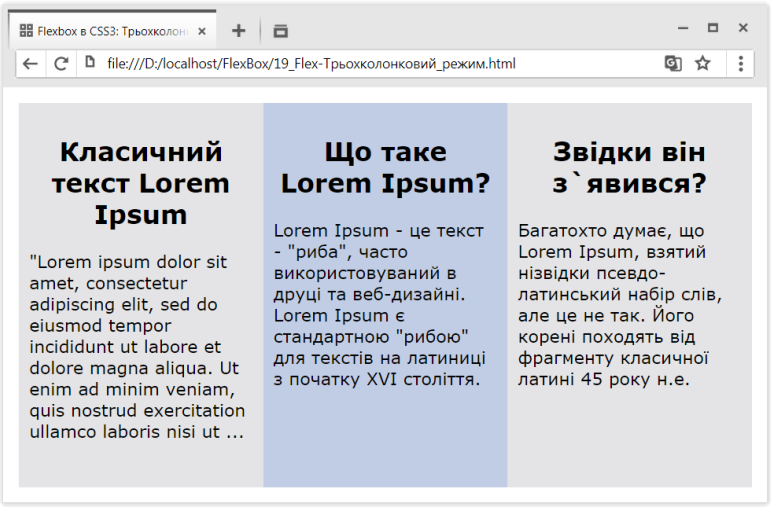


Рис. 3.68. Сторінка з трьома колонками

Подібним чином можна додати і більшу кількість стовпців. Але в даному випадку, за замовчуванням, стовпці мають однакову ширину. Але що робити, якщо **один із стовпців** (як правило, центральний) **повинен мати ширину більшу, ніж в інших?** Для цього додамо в стилі сторінки таке правило:

.flex-item:first-child { flex: 0 0 50%; }

Тоді перший елемент завжди буде займати 50% простору контейнера.

3.7.10. Макет сторінки на Flexbox

Розглянемо створення **стандартного макету сторінки**, що складається з шапки, футера і центральної частини, в якій є три стовпці: основний вміст і два сайдбари.

Схематично сторінка виглядає так:

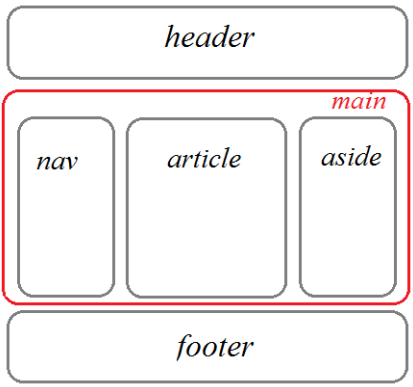


Рис. 3.69. Схема макету веб-сторінки

Визначимо код веб-сторінки:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Макет сторінки на Flexbox </title>
  <style>
   *{ box-sizing: border-box; }
```

```
  h2 { text-align:center; color: white;}
  h3 { text-align:center; color: black; margin-top:40px;}
  html, body { padding: 0; margin: 0;
                font-family: verdana, arial, sans-serif; }
  body { display: flex; flex-direction: column;
         font-size: 1.2em;
        padding: 1em; }
  main { display: flex; flex-direction: column; }
  article {  flex: 2 2 12em;
             padding: 1.2em;
            background-color: #E4E4E6;
            color:black; }
  nav, aside { flex: 1; background-color: #CBC6D9; }
  nav { order: -1; }
  header, footer { flex: 0 0 5em; background-color: #6089AE; }
  @media screen and (min-width: 600px) {
    body{  min-height: 100vh;
        /* vh еквівалентно 1% висоти вікна браузера */ }
    main { flex-direction: row; min-height: 100%;
           flex: 1 1 auto; }
    }
 </style>
</head>
<body>
  <header>  <h2>Header</h2>  </header>
  <main>
   <article>
     <h1>Що таке Lorem Ipsum?</h1>
     <p>Lorem Ipsum – це … </p>
   </article>
   <nav>  <h3>Navigation</h3>  </nav>
   <aside>  <h3>Sidebar</h3>  </aside>
  </main>
  <footer>  <h2>Footer</h2>  </footer>
 </body>
</html>
```

Створений макет має вигляд:

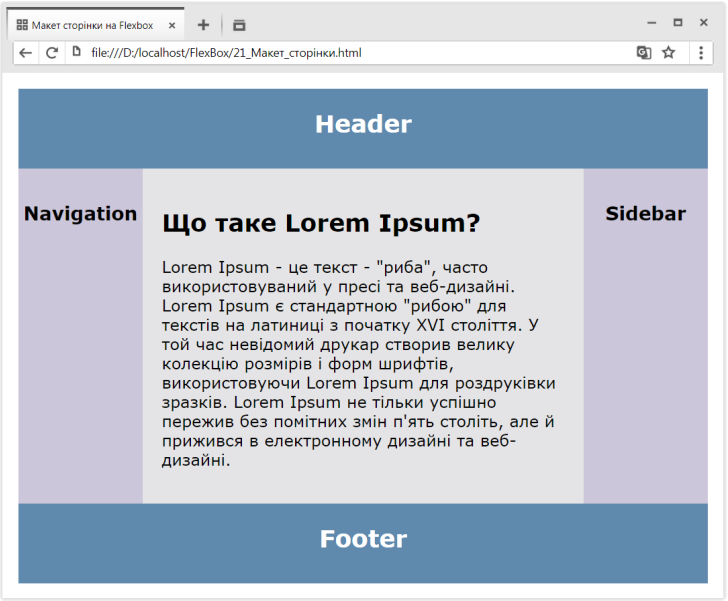


Рис. 3.70. Макет сторінки на FlexBox

Отже, ***flex*-контейнером верхнього рівня є елемент *body***. Його *flex*-елементами є ***header***, ***main*** і ***footer***. *Body* розташовує всі свої елементи зверху вниз у стовпчик. При ширині **від 600px і більше** для заповнення всього простору браузера в *body* встановлюється стиль *height: 100vh;*.

Елементи *header* і *footer* аналогічні. Їх властивість *flex: 0 0 5em;* вказує, що при будь-якій зміні контейнера ці елементи будуть мати розмір 5em.

Більш складний **елемент *main***, який визначає основний вміст. При цьому, будучи *flex*-елементом, він також є *flex*-контейнером для вкладених елементів і управляє їх позиціонуванням. При ширині браузера **до 600px** він розташовує елементи в стовпчик, що зручно на мобільних пристроях.

При ширині **від 600px** вкладені елементи *nav*, *article* і *aside* розташовуються у вигляді рядка. І оскільки при такій ширині браузера батьківський елемент *body* заповнює по висоті весь простір браузера, то для заповнення всієї висоти контейнера

body при його зміні в елемента *main* встановлюється властивість *flex: 1 1 auto;*.

Для вкладених в *main flex*-елементів елемент навігації *nav* і елемент сайдбару *aside* матимуть однакові розміри при масштабуванні контейнера. А елемент *article*, що містить основний вміст, буде відповідно більшим. При цьому хоча *nav* визначено після елемента *article*, але завдяки заданню властивості *order: -1* блок навігації буде розміщуватися до блоку *article*.

3.8. Grid Layout

3.8.1. Що таке Grid Layout. Grid Container

Grid Layout являє собою спеціальний модуль CSS3, який дозволяє позиціонувати елементи у вигляді сітки або таблиці. Як і Flexbox, Grid Layout забезпечує гнучкий підхід до компонування елементів. Але якщо Flexbox розміщує вкладені елементи в одному напрямку: або по горизонталі, або по вертикалі, то Grid позиціонує елементи одночасно в двох напрямках – у вигляді рядків і стовпців, утворюючи тим самим таблицю.

Повністю специфікацію модуля Grid Layout можна подивитися на сайті www.w3.org/TR/css-grid-1/.

Підтримка браузерами. При використанні Grid Layout слід враховувати, що тільки у відносно недавно випущених браузерах впроваджено підтримку цього модуля. Наведемо список версій браузерів, починаючи з яких була впроваджена повноцінна підтримка Grid Layout: Google Chrome – з версії 57, Mozilla Firefox – з версії 52, Opera – з версії 44, Safari – з версії 10.1, iOS Safari – з версії 10.3. Як можна помітити, більшість цих версій браузерів вийшли на початку 2017 року, тобто на їх старші версії розраховувати не доводиться. Крім того, IE (починаючи з версії 10) та Microsoft Edge має лише часткову підтримку модуля, а Android Browser, Opera Mini, UC Browser зовсім її не мають.

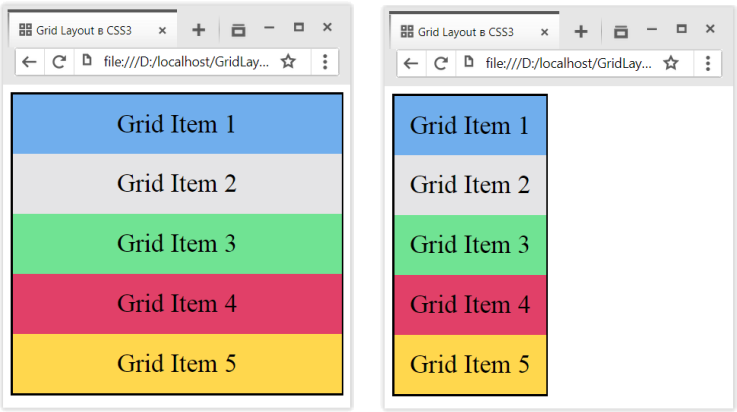
Створення grid-контейнера. Основою компоновки Grid Layout є grid-контейнер (grid-container), всередині якого розміщуються елементи. Для створення grid-контейнера потрібно присвоїти його стильовій **властивості *display*** одне з двох значень: ***grid*** або ***inline-grid***.

Значення *grid* визначає контейнер як блоковий елемент, а **значення *inline-grid*** визначає елемент як рядковий (*inline*).

Наприклад, визначимо найпростішу веб-сторінку, яка застосовує Grid Layout. Для цього у розділі стилів створимо *grid*-контейнер та *grid*-елементи. Для контейнера *grid-container* встановимо властивість ***display: grid***. У ньому розташуємо п'ять grid-елементів.

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Grid Layout в CSS3</title>
  <style>
    .grid-container { display: grid;
                      border:solid 2px black; }
    .grid-item {  /*стильові налаштування для елемента*/
         text-align:center;  font-size: 1.7em;   padding: 0.6em; }
     /*задання кольорів фону елементів */
     .color1 { background-color: #70AEED;}
     .color2 {…}  .color3 {…}  .color4 {…}  .color5 {…}
  </style>
 </head>
 <body>
  <div class="grid-container">
     <div class="grid-item color1">Grid Item 1</div>
     <div class="grid-item color2">Grid Item 2</div>
     <div class="grid-item color3">Grid Item 3</div>
     <div class="grid-item color4">Grid Item 4</div>
     <div class="grid-item color5">Grid Item 5</div>
  </div>
 </body>
</html>
```

У цьому випадку grid-контейнер займає весь простір (рис. 3.71а). Але якщо для контейнера задати "*display: inline-grid;*", то він займатиме тільки той простір, який необхідний для розміщення елементів (рис. 3.71б):



а) *б)*

Рис. 3.71. Властивість display:
а – значення *grid; б* – значення *inline-grid.*

3.8.2. Рядки та стовпці

Грід утворює сітку з рядків і стовпців, на перетині яких утворюються комірки. Для задання рядків і стовпців в Grid Layout використовуються такі **властивості CSS3**:

- ***grid-template-columns***: визначає стовпці;
- ***grid-template-rows***: визначає рядки.

Для визначення стовпців *grid*-контейнера використовується стильова властивість *grid-template-columns*. Значенням властивості *grid-template-columns* є ширина стовпців. Скільки треба мати в грід-контейнері стовпців, стільки і слід вказати значень у цій властивості.

Наприклад, визначимо грід-контейнер із двома стовпцями, ширина кожного з яких *8em*:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Grid Layout в CSS3</title>
  <style>
   .grid-container { display: inline-grid;
                     grid-template-columns: 8em 8em;
```

```
                    border: 2px solid  black; }
    .grid-item { /*стильові налаштування для елемента*/
        text-align:center;  font-size: 1.7em;  padding: 0.6em; }
     /*задання кольорів фону елементів */
    .color1 { background-color: #70AEED; }
    .color2 {…}  .color3 {…}  .color4 {…}  .color5 {…}
  </style>
 </head>
 <body>
   <div class="grid-container">
        <div class="grid-item color1">Grid Item 1</div>
        <div class="grid-item color2">Grid Item 2</div>
        <div class="grid-item color3">Grid Item 3</div>
        <div class="grid-item color4">Grid Item 4</div>
        <div class="grid-item color5">Grid Item 5</div>
   </div>
 </body>
</html>
```

Такий грід-контейнер, наповнений елементами, має вигляд:

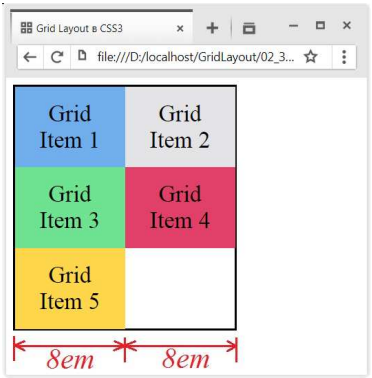


Рис. 3.72. Грід-контейнер із двома стовпцями

Відповідно, якщо потрібно, щоб у грід-контейнері було три стовпці, то перераховують три значення, розділених пробілом:

grid-template-columns: 8em 7em 8em;

Якщо елементів більше, ніж комірок, то, за замовчуванням, для їх розміщення створюються нові рядки.

Визначення рядків багато в чому аналогічне визначенню стовпців. Для цього у грід-контейнері необхідно встановити властивість *grid-template-rows*, яка задає кількість та розміри рядків. Наприклад, висота першого рядка становить *4em*, другого – *5em*, третього і четвертого – *4em*:

```
<style>
 .grid-container { display: grid;
       grid-template-columns: 9em 9em 9em;
       grid-template-rows: 4em 5em 4em 4em;
      border: 2px solid black; }
 .grid-item { /*стильові налаштування для елемента*/ }
 …
 </style>
</head>
<body>
    <div class="grid-container">
       <div class="grid-item color1">Grid Item 1</div>
       <div class="grid-item color2">Grid Item 2</div>
       <div class="grid-item color3">Grid Item 3</div>
       <div class="grid-item color4">Grid Item 4</div>
       <div class="grid-item color5">Grid Item 5</div>
    </div>
 </body>
</html>
```

Такий *grid*-контейнер, наповнений елементами, має вигляд:

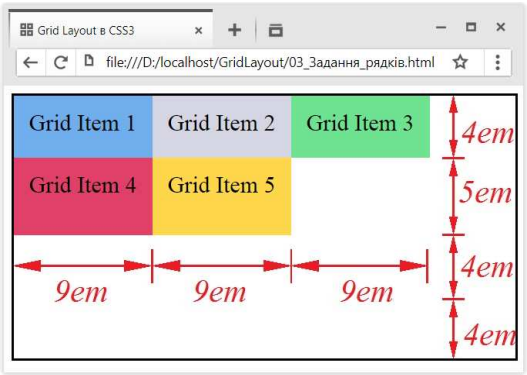


Рис. 3.73. Грід-контейнер із чотирма рядками

Оскільки комірок грід-контейнера більше, ніж елементів, то останні два рядки залишаються порожніми.

У той же час, якщо елементів більше, ніж комірок грід-контейнера, то утворюються додаткові рядки (як у випадку зі стовпцями), при цьому висота доданого рядка буде обчислюватися автоматично.

3.8.3. Функція repeat та властивість grid

1. Повторення рядків і стовпців. Якщо стовпців і/або рядків багато і вони мають однакові розміри, то є сенс використовувати спеціальну **функцію *repeat()***, яка дозволяє скорочено визначати такі рядки і стовпці.

Наприклад, є визначення рядків і стовпців у *grid*-контейнері:

```
grid-template-columns: 8em 8em 8em;
grid-template-rows: 5em 5em 5em 5em;
```

де відбувається повторення одних і тих же розмірів – *8em* і *5em* для задання ширини стовпців і висоти рядків. Тому перепишемо стилі, використовуючи **функцію *repeat,* де** перший параметр вказує кількість повторень, а другий – задає розмір, що повторюється:

```
grid-template-columns: repeat(3, 8em);
    /*3 стовпці шириною в 8em */
grid-template-rows: repeat(4, 5em);
    /*4 рядки висотою по 5em */
```

Можна задавати повторення кількох стовпців і рядків:

```
grid-template-columns: repeat(2, 7em 8em); /* буде створено 4 стовпці: двічі будуть повторюватися стовпці з шириною 7em і 8em */

grid-template-rows: 6em repeat(3, 5em); /* буде створено 4 рядки: перший матиме висоту 6em, а наступні три - 5em */
```

2. Властивість grid об'єднує властивості *grid-template-rows* і *grid-template-columns*, тобто дозволяє разом визначити задання рядків і стовпців у такому форматі:

```
grid: <grid-template-rows> / <grid-template-columns>;
```

де в кутових дужках зазначається значення, вказаної в них властивості. Наприклад, є таке визначення структури рядків та стовпців *grid*-контейнера:

grid-template-columns: 8em 8em 8em;
grid-template-rows: 5em 5em 5em 5em;
Цей запис можна скоротити наступним чином:
grid: 5em 5em 5em 5em / 8em 8em 8em;
Знову ж використовуючи функцію *repeat()*, можна ще більше скоротити визначення структури грід-контейнера:
grid: repeat(4, 5em) / repeat(3, 8em);

3.8.4. Розміри рядків і стовпців

1. Фіксовані розміри. У прикладах, які були розглянуті в попередніх темах, ширина стовпців і висота рядків встановлювалися на підставі **фіксованих значень**, які передаються властивостям *grid-template-columns* і *grid-template-rows*. Для задання розмірів можна використовувати найрізноманітніші одиниці виміру, які доступні в CSS (*px*, *em*, *rem*, %), наприклад:

```
.grid-container { display: grid;
   grid-template-columns: repeat(3, 200px);
   grid-template-rows: repeat(3, 4.5em);
   border: solid 2px black;
}
```

Одиниця виміру *rem* задає розмір відносно розміру шрифта елемента <*html*>. Як правило, браузер визначає для цього елемента прийнятний (reasonable) розмір шрифту за замовчуванням, який ми, звичайно, можемо перевизначити і використовувати для задання розмірів шрифтів всередині тега <*html*> відносно нього ж.

Наприклад, визначимо розмір шрифту:

```
<style>
   html { font-size: 15px; }
   div { font-size: 0.8rem; }
</style>
```

У такому випадку на веб-сторінці у всіх елементах *div* розмір шрифту становитиме *15×0.8=12px*.

2. Крім точних розмірів, можна задати **автоматичні розміри** за допомогою ключового слова *auto*. Тоді ширина стовпців і висота рядків обчислюється, виходячи з розміру їх вмісту.

Приклад. Нехай у грід-контейнері задано три рядки і три стовпці. Перший стовпець має фіксовану ширину *8em*, а другий і третій стовпці – автоматичну. Крім того, перший і третій рядки мають автоматичну висоту, а другий рядок – фіксовану:

```
.grid-container { display: grid;
   grid-template-columns: 8em auto auto;
   grid-template-rows: auto 4.5em auto;
   border: solid 2px black;
}
```

3. Для задання **пропорційних розмірів** застосовується спеціальна одиниця вимірювання *fr*. Вона являє собою частину простору (*fraction*), який відводиться для даного стовпця або рядка. Значення *fr* ще називають ***flex*-фактором** (*flex factor*).

Обчислення пропорційних розмірів здійснюється за формулою:

*flex-фактор * доступний_простір / сума всіх flex-факторів.*

При цьому під доступним простором розуміється весь простір грід-контейнера, за винятком фіксованих значень рядків і стовпців.

Наприклад, нехай визначено грід-контейнер та налаштування стилів сторінки:

```
<style>
   *{ box-sizing: border-box; }
   html, body { margin:0; padding:0; }
   .grid-container { display: grid; height: 100vh;
      grid-template-columns: 8em 2fr 1fr;
      grid-template-rows: 1fr 4.5em 1fr;
      border: solid 2px black;
   }
</style>
```

У цьому випадку, є три стовпці шириною *8em*, *2fr*, *1fr*. Тому ширина другого стовпця буде обчислюватися за формулою

2×(ширина_гріду - 8em) / (2 + 1).

Ширина третього стовпця обчислюватиметься так:

1×(ширина_гріду - 8em) / (2 + 1).

Якщо перший стовпець фіксований із шириною *8em*, то ширина другого і третього стовпців буде залежати від ширини грід-контейнера, і автоматично масштабуватиметься при його зміні.

Відносно рядків все аналогічно.

3.8.5. Відступи між стовпцями та рядками

Для створення відступів між стовпцями та рядками використовуються **властивості *grid-column-gap* і *grid-row-gap*** відповідно.

Наприклад, визначимо грід-контейнер із відступами між рядками 10 пікселів і між стовпцями – 20, при цьому перевизначимо задання розмірів на сторінці та відступи за замовчуванням:

```
<style>
  *{ box-sizing: border-box; }
  html, body { margin:0; padding:0; }
  .grid-container {
      display: grid; height: 100vh;
      grid-template-columns: repeat(3, 1fr);
      grid-template-rows: repeat(3, 1fr);
      grid-column-gap: 20px; grid-row-gap: 10px;
  }
</style>
```

Якщо значення властивостей *grid-column-gap* і *grid-row-gap* збігаються, то замість них можна визначити одну **властивість** *grid-gap*, яка задає обидва відступи:

```
<style>
  .grid-container {
    display: grid; height: 100vh;
    grid-template-columns: repeat(3, 1fr);
    grid-template-rows: repeat(3, 1fr);
    grid-gap: 10px;
  }
</style>
```

3.8.6. Позиціонування елементів

Грід являє собою набір комірок, які утворюються на перетині стовпців і рядків. Але самі рядки і стовпці утворюються за допомогою ***grid*-ліній**, які розсікають грід по вертикалі і горизонталі:

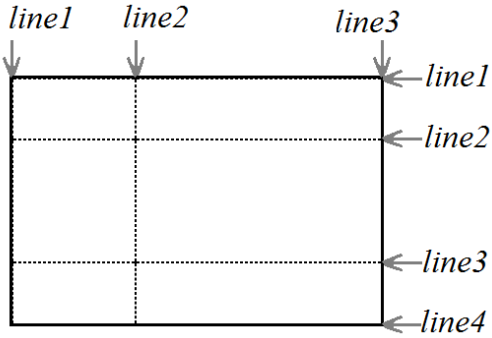


Рис. 3.74. Розмітка грід-контейнера грід-лініями

За замовчуванням, **кожен елемент у гріді позиціонується як одна комірка**. Елементи, перераховані в грід-контейнері, розташовуються на html-сторінці рядками по горизонталі в порядку їх розміщення в коді.

За допомогою **набору властивостей**, які застосовуються до елементів грід-контейнера, можна організувати розтягування (зміну розміру) комірок у контейнері:

- ***grid-row-start*** задає початкову горизонтальну *grid*-лінію, з якої починається елемент;
- ***grid-row-end*** вказує, до якої горизонтальної *grid*-лінії буде розтягуватися елемент;
- ***grid-column-start*** задає початкову вертикальну *grid*-лінію, з якої починається елемент;
- ***grid-column-end*** вказує, до якої вертикальної *grid*-лінії буде розтягуватися елемент.

1. Розтягування елемента на кілька стовпців (по горизонталі)

Наприклад, визначимо стилі для грід-контейнера, грід-елемента, а також окремий стиль, що буде застосований до другого елемента контейнера, щоб розширити його на два наступні стовпці:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
```

```
<meta name="viewport" content="width=device-width" />
<title>Grid Layout в CSS3</title>
<style>
 .grid-container { display: grid; border: solid 2px black;
   grid-template-columns: repeat(4, 1fr); /*4 стовпці */
   grid-template-rows: repeat(3, 4em);  /*3 рядки */
 }
 .special-item { grid-column-start:2; grid-column-end: 5; }
 .grid-item { /*стиль грід-елемента */
     text-align:center;  font-size: 1.4em;
     padding: 0.6em; }
 /* задання кольору фону грід-елементів */
 .color1 { background-color: #70AEED; }
 .color2 {…}  .color3 {…}
 .color4 {…}  .color5 {…}
</style>
</head>
<body>
 <div class="grid-container">
   <div class="grid-item color1">Grid Item 1</div>
   <div class="grid-item color2 special-item">
         Grid Item 2</div>
   <div class="grid-item color3">Grid Item 3</div>
   <div class="grid-item color4">Grid Item 4</div>
   <div class="grid-item color5">Grid Item 5</div>
   <div class="grid-item color1">Grid Item 6</div>
   <div class="grid-item color4">Grid Item 7</div>
 </div>
</body>
</html>
```

У контейнері визначено чотири стовпці і три рядки, причому другому елементу присвоєно спеціальний клас *special-item*, який розташовується, починаючи з другої *grid*-лінії або 2-го стовпця (*grid-column-start: 2)* до 5-ї вертикальної *grid*-лінії (*grid-column-end: 5*).

Результат має вигляд:

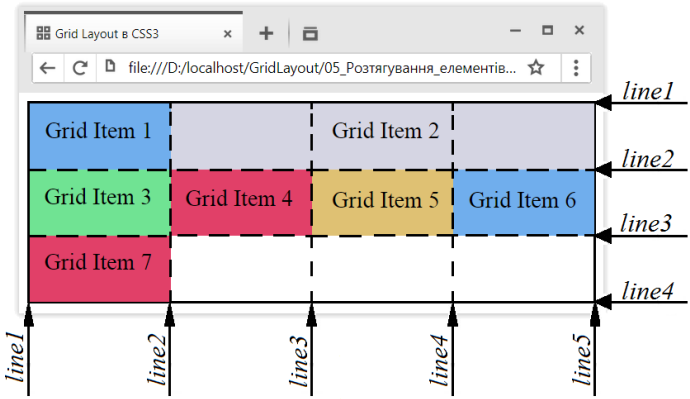


Рис. 3.75. Розтягування комірок у грід-контейнері

Другий елемент необов'язково повинен починатися з другого стовпця, це може бути будь-який інший: і 1-й, і 3-й і т.д. Наприклад, якщо розмістити другий елемент, починаючи з 3-го стовпця, то на місці другого елемента залишиться порожнє місце:

.special-item{ grid-column-start: 3; grid-column-end: 5; }

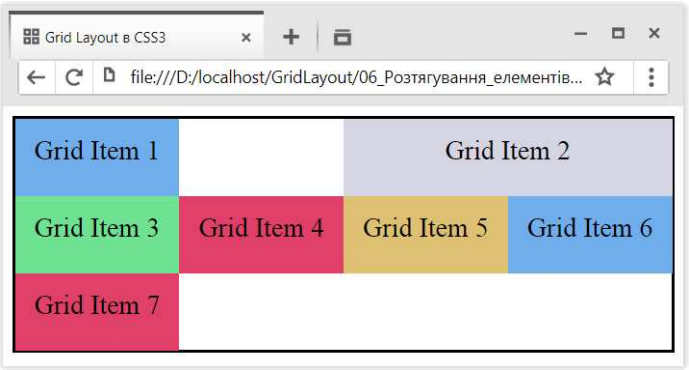


Рис. 3.76. Розтягування комірок у грід-контейнері

Якщо для другого елемента встановити початок із першого стовпця, то другий елемент буде перенесений на наступний рядок, і таким чином, він буде починатися з першого стовпця.

Замість використання двох вищевказаних властивостей можна використовувати одну **властивість *grid-column***, яка приймає значення *grid-column-start* і *grid-column-end* через слеш:

grid-column: <grid-column-start> / <grid-column-end>;

тобто дозволяє скоротити визначення стилю класу *special-item*:

.special-item { grid-column: 3/5; }

2. Розтягування елемента на кілька рядків (по вертикалі)

Аналогічно, за допомогою властивостей *grid-row-start* і *grid-row-end* можна задати позиціонування елемента на кілька рядків. Змінимо клас *special-item* наступним чином:

.special-item{ grid-column-start:2;
grid-row-start: 1; grid-row-end: 3; }

У цьому випадку другий елемент позиціонується в другому стовпці й розтягується від першої горизонтальної грід-лінії до третьої:

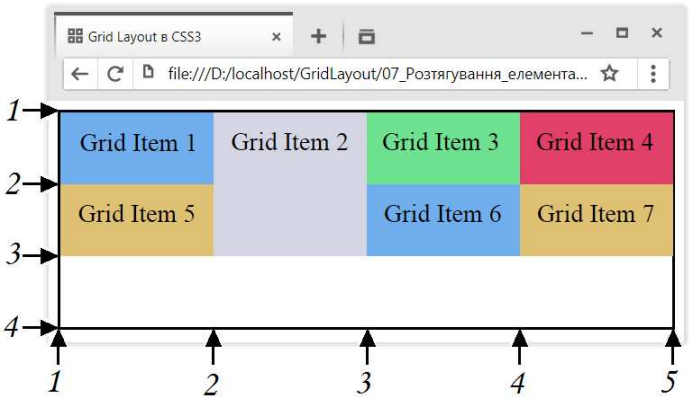


Рис. 3.77. Розтягування елемента на кілька рядків

Замість використання пари властивостей *grid-row-start* і *grid-row-end* можна використовувати одну загальну **властивість *grid-row***:

grid-row: <grid-row-start> / <grid-row-end>;

Отже, стиль *special-item* можна записати наступним чином:

.special-item { grid-column-start:2; grid-row: 1 / 3; }

3. За допомогою **спеціального слова *span*** можна задати розтягування елемента на кілька комірок. Після слова *span* вказується, на яку кількість комірок потрібно розтягнути елемент:

.special-item { grid-row: 1 / span 2; grid-column: 2 / span 2; }

Рис. 3.78. Використанням ключового слова *span*

Елемент поміщається в комірку, яка знаходиться на перетині першого рядка і другого стовпця, і розтягується на два рядки і на два стовпці в напрямку вниз і вправо, де першим рядком і стовпцем комірки є, відповідно, рядок і стовпець її розташування.

4. Властивість *grid-area* об'єднує властивості *grid-column* і *grid-row*, дозволяючи скоротити їх запис:

grid-area: <grid-row-start> / <grid-column-start> /
<grid-row-end> / <grid-column-end>;

Наприклад, змінимо стиль класу special-item:

.special-item{ grid-area: 1 / 2 / 3 / 4; }

Результат буде тим самим, що й на рис. 3.78.

3.8.7. Накладання елементів

Маніпулюючи розміщенням елементів, можна здійснити їх накладання, створюючи свого роду прошарки елементів. Наприклад, визначимо стилі веб-сторінки:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
```

```
<meta name="viewport" content="width=device-width" />
<title>Grid Layout в CSS3</title>
<style>
 *{ box-sizing: border-box; }
 html, body{ margin: 0px; padding:0px; }
 .grid-container { height:100vh;
    display: grid; grid-gap: 10px;
    grid-template-rows: repeat(3, 1fr);
    grid-template-columns: repeat(3, 1fr);
    border: solid 3px black;  }
 .grid-item { /*стиль грід-елемента */
     text-align:center;  font-size: 1.6em; padding: 0.5em; }
 .item1 { grid-area: 1 / 1 / 3 / 4; }
 .item2 { grid-area: 1 / 1 / 2 / 2; }
 .item3 { grid-area: 1 / 3 / 2 / 4; }
 .item4 { grid-area: 2 / 1 / 3 / 2; }
 .item5 { grid-area: 2 / 2 / 3 / 3; }
 .item6 { grid-area: 2 / 3 / 3 / 4; }
 .item7 { grid-area: 3 / 1 / 4 / 2;  }
 /*задання кольору фону грід-елемента */
 .color1 { background-color: #70AEED; }
 .color2 {…} .color3 {…} .color4 {…} .color5 {…}
</style>
</head>
<body>
  <div class="grid-container">
    <div class="grid-item item1 color2">Grid Item 2</div>
    <div class="grid-item item2 color1">Grid Item 1</div>
    <div class="grid-item item3 color3">Grid Item 3</div>
    <div class="grid-item item4 color4">Grid Item 4</div>
    <div class="grid-item item5 color1">Grid Item 5</div>
    <div class="grid-item item6 color4">Grid Item 6</div>
    <div class="grid-item item7 color5">Grid Item 7</div>
  </div>
</body>
</html>
```

Результуюча сторінка має вигляд:

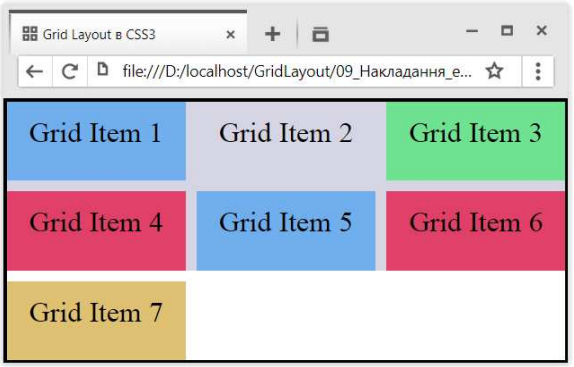


Рис. 3.79. Накладання елементів

Перший у грід-контейнері елемент (з класом *“grid-item item1 color2”*) наноситься на веб-сторінку першим, тому наступні п’ять елементів накладаються на нього. Таким чином, отримуємо, що перший елемент візуально виглядає фоном для інших п’яти елементів.

При необхідності шар із першого елемента можна, навпаки, перенести ближче до користувача, накривши інші елементи. Для цього потрібно задати для нього невід’ємне значення **властивості *z-index***:

.item1 { grid-area: 1 / 1 / 3 / 4; z-index: 1; opacity: 0.8; }

Рис. 3.80. Накладання елементів із заданням властивості *z-index*

Властивість ***z-index*** дозволяє керувати порядком накладання елементів. Елементи з більшим значенням властивості *z-index* будуть відображатися поверх елементів з меншим значенням цієї властивості.

Значення *z-index*, за замовчуванням *"auto"*: порядок елементів у цьому випадку будується автоматично, виходячи з розташування в HTML-коді і приналежності до батьківського контейнера. При цьому елементи з невід'ємним значенням цієї властивості відображатимуться поверх елементів зі значенням *"auto"*, а елементи зі значенням *"auto"* – поверх елементів із від'ємним значенням властивості *z-index*.

3.8.8. Напрямок та порядок елементів

За замовчуванням, усі елементи розташовуються по порядку горизонтально. Якщо в рядку не залишилось місця, то елементи переносяться на наступний рядок. Але за допомогою **властивості *grid-auto-flow*** можна змінити напрям розташування елементів. Ця властивість приймає два значення:

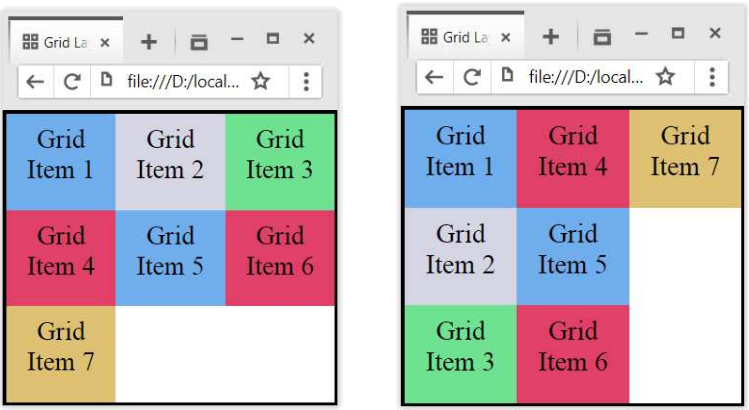
- ***row***: значення, за замовчуванням, елементи розташовуються в рядок один за одним, якщо місця в рядку не вистачає, елементи переносяться на наступний рядок;
- ***column***: елементи розташовуються в стовпчик, якщо місця в стовпці не вистачає, то наступні елементи розміщуються у наступному стовпці.

Використаємо цю властивість у стилі грід-контейнера:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Grid Layout в CSS3</title>
  <style>
   *{ box-sizing: border-box; }
   html, body{ margin:0px; padding:0px; }
   .grid-container { height: 100vh;
      display: grid; grid-auto-flow: row;
      grid-template-rows: repeat(3, 1fr);
      grid-template-columns: repeat(3, 1fr);
```

```
      border: solid 3px black;
    }
    .grid-item { text-align:center; font-size: 1.4em;
                 padding: 0.4em; }
    .color1 { background-color: #70AEED;}
    .color2 {…}   .color3 {…}
    .color4 {…}   .color5 {…}
  </style>
 </head>
 <body>
   <div class="grid-container">
     <div class="grid-item color1">Grid Item 1</div>
     <div class="grid-item color2">Grid Item 2</div>
     <div class="grid-item color3">Grid Item 3</div>
     <div class="grid-item color4">Grid Item 4</div>
     <div class="grid-item color1">Grid Item 5</div>
     <div class="grid-item color4">Grid Item 6</div>
     <div class="grid-item color5">Grid Item 7</div>
   </div>
 </body>
</html>
```

Проілюструємо значення **властивості *grid-auto-flow*:**



а) *б)*

Рис. 3.81. Властивість *grid-auto-flow:*
а – значення *row*; *б* – значення *column*.

За замовчуванням, елементи на веб-сторінці розташовуються один за одним у порядку їх визначення в html-розмітці. Але цей порядок можна змінити, використовуючи **властивість *order*.**

Властивість *order* дозволяє перевизначити порядок розташування елементів. Властивість визначається для *grid*-елемента. Значенням властивості є довільне ціле число, в тому числі й від'ємне. За замовчуванням, для кожного елемента в гріді ця властивість має значення 0.

Елементи в грід-контейнері групуються за значенням властивості *order*. Групи розташовуються за зростанням їх номера. Елементи всередині групи (ті, які мають однакове значення *order*) розміщуються в контейнері в порядку їх розташування в html-розмітці.

Приклад. Нехай є стилі грід-контейнера та грід-елемента, визначимо ще дві групи грід-елементів зі значеннями властивості *order*: -1 і 1. Крім того, елементи, для яких ця властивість не задана, утворюють групу з номером 0 (значення властивості *order* за замовчуванням):

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Grid Layout в CSS3</title>
  <style>
   *{ box-sizing: border-box; }
  html, body{ margin:0; padding:0; }
  .grid-container {
      display: grid; height:100vh;
      grid-template-rows: repeat(3, 1fr);
      grid-template-columns: repeat(3, 1fr);
      border:solid 3px black; }
  .grid-item { /*стиль грід-елемента */
       text-align:center;  font-size: 1.5em;
       padding: 0.4em; }
  .first-item { order: -1; }
  .last-item { order: 1; }
```

```
  /*задання кольору фону грід-елементів */
  .color1 {background-color: #70AEED;}
  .color2 {…} .color3 {…} .color4 {…} .color5 {…}
 </style>
</head>
<body>
 <div class="grid-container">
   <div class="grid-item color2">Grid Item 1</div>
   <div class="grid-item color5 last-item">Grid Item 2</div>
   <div class="grid-item color2 last-item">Grid Item 3</div>
   <div class="grid-item color3">Grid Item 4</div>
   <div class="grid-item color3">Grid Item 5</div>
   <div class="grid-item color4">Grid Item 6</div>
   <div class="grid-item color1 first-item">Grid Item 7</div>
 </div>
</body>
</html>
```

У результаті, в грід-контейнері першим розташовуватиметься елемент 7 з класом *first-item*. Наступними є елементи 1,4,5,6, для яких не задано властивість *order*, вони утворюють групу 0 і виводяться в порядку їх розміщення в html-коді. Елементи 2 і 3 з класом *last-item* мають порядок 1, тому вони будуть розташовуватися після інших елементів, у яких порядок дорівнює 0 або менше.

Отже, отримуємо

Рис. 3.82. Властивість *order*

3.8.9. Іменовані grid-лінії

У Grid Layout є можливість дати **назву кожній лінії гріду**, присвоївши їй ім'я у квадратних дужках і, використовуючи це ім'я, позиціонувати елементи.

При іменуванні грід-ліній між ними вказується ширина стовпця або висота рядка, які знаходяться між цими лініями. Потім, використовуючи ці назви, можна позиціонувати елементи між цими лініями.

Наприклад, визначимо стилі грід-контейнера та грід-елементів веб-сторінки:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Grid Layout в CSS3</title>
  <style>
   *{ box-sizing: border-box; }
  html, body { margin:0px; padding:0px; }
  .grid-container {
      display: grid; height:100vh;
      grid-template-columns: [col1start] 1fr [col1end] 10px
                             [col2start] 1fr [col2end] 10px
                             [col3start] 1fr [col3end];
      grid-template-rows: [row1start] 1fr [row1end] 10px
                          [row2start] 1fr [row2end];
  }
  .grid-item {  /*стиль грід-елемента */
      background-color: #C1CDE5;
      text-align:center;
      font-size: 1.7em;
      padding: 0.5em; }
  .special-item {
      grid-column: col1start / col2end;
      grid-row: row1start;
      background-color: #D7B8CA; }
  .item1 {  grid-column: col3start / col3end;
            grid-row: row1start; }
```

```
    .item2 {  grid-column: col1start / col1end;
              grid-row: row2start; }
    .item3 {  grid-column: col2start / col2end;
              grid-row: row2start; }
    .item4 {  grid-column: col3start / col3end;
              grid-row: row2start; }
  </style>
 </head>
 <body>
  <div class="grid-container">
     <div class="grid-item special-item">Special Item</div>
     <div class="grid-item item1"> Grid Item 1</div>
     <div class="grid-item item2"> Grid Item 2</div>
     <div class="grid-item item3"> Grid Item 3</div>
     <div class="grid-item item4"> Grid Item 4</div>
  </div>
 </body>
</html>
```

У даному випадку, елемент з класом *special-item* починається від вертикальної лінії *col1start* і розтягується до вертикальної лінії *col2end*. Також він починається від горизонтальної лінії *row1start* і, оскільки кінцева горизонтальна лінія не вказана, то елемент займає тільки один рядок.

У результаті отримуємо

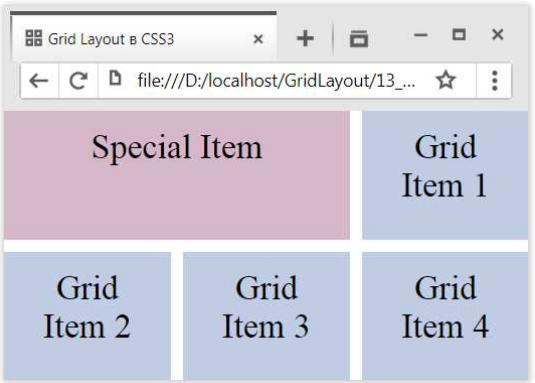


Рис. 3.83. Використання іменованих грід-ліній

Розглянемо **приклад створення найпростішого макету веб-сторінки на основі Grid Layout із використанням іменованих грід-ліній**:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Макет сторінки на Grid Layout</title>
  <style>
    *{ box-sizing: border-box; }
    html, body { padding: 0px; margin: 0 px; }
    h1 { font-size: 2.0em; text-align: center; }
    .grid-container {
        height:100vh;
        display: grid;
        grid-template-rows: 5em 10px 1fr;
        grid-template-columns: [mainstart] 1fr [mainend] 10px
                               [sidebarestart] 25% [sidebarend];
    }
    .grid-item-content {
     grid-row: 3 / 4;  grid-column: mainstart / mainend;
     background-color: #D7B8CA;
    }
    .grid-item-sidebar {
     grid-row: 3 / 4;  grid-column: sidebarestart / sidebarend;
     background-color: #DDDDD7;
    }
    .grid-item-header {
     grid-row: 1 / 2;  grid-column: mainstart / sidebarend;
     background-color: #C1CDE5;
    }
  </style>
 </head>
 <body>
  <main class="grid-container">
   <header class="grid-item-header">
     <h1>Header</h1>
```

```
    </header>
    <article class="grid-item-content">
      <h1>Main Content</h1>
    </article>
    <aside class="grid-item-sidebar">
      <h1>Sidebar</h1>
    </aside>
  </main>
 </body>
</html>
```

Макет має вигляд:

Рис. 3.84. Макет сторінки на Grid Layout

3.8.10. Іменовані grid-лінії та функція repeat

За допомогою раніше розглянутої **функції *repeat*** ми можемо розтиражувати стовпці і рядки, які створюються між іменованими grid-лініями.

Розглянемо приклад:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Grid Layout в CSS3: функція repeat</title>
```

```
 <style>
  *{ box-sizing: border-box;}
  html, body{ margin:0px; padding:0px; }
  .grid-container {
    height:100vh;
    display: grid;
    grid-template-columns: 10px
                    repeat(3, [column] 1fr [colgutter] 10px);
    grid-template-rows: 10px
                   repeat(2, [row] 1fr [rowgutter] 10px);
  }
  .grid-item {
    background-color: #C1CDE5; padding: 0.4em;
    font-size: 1.7em; text-align: center;
  }
  .special-item {
    grid-column: column 2;   /*елемент в другому стовпці,
                             що поч. з грід-лінії column*/
    grid-row: row 1;   /*елемент в першому рядку,
                             що поч. з грід-лінії row*/
    background-color: #DEBFAF;
  }
  .item1 { grid-column: column 1; grid-row: row 1; }
  .item2 { grid-column: column 3; grid-row: row 1; }
  .item3 { grid-column: column 1; grid-row: row 2; }
  .item4 { grid-column: column 2; grid-row: row 2; }
 </style>
</head>
<body>
 <div class="grid-container">
  <div class="grid-item special-item">Special-item</div>
  <div class="grid-item item1">Grid Item1</div>
  <div class="grid-item item2">Grid Item2</div>
  <div class="grid-item item3">Grid Item3</div>
  <div class="grid-item item4">Grid Item4</div>
 </div>
</body>
</html>
```

У визначенні стовпців властивістю *grid-template-columns:*
grid-template-columns:

10px repeat(3, [column] 1fr [colgutter] 10px);

перший стовпець буде мати ширину 10 пікселів. Потім відбувається тиражування стовпців за допомогою функції *repeat*. Вона створює підряд три копії двох стовпців: перший стовпець має ширину *1fr*, тобто має пропорційні розміри і розташовується між grid-лініями "*column*" і "*colgutter*". Після grid-лінії "*colgutter*" йде ще один стовпець шириною 10 пікселів, і ці два стовпці будуть повторюватися три рази. Таким чином, всього в гріді буде 7 стовпців.

З рядками аналогічно, хоча створюється 5 рядків за допомогою grid-ліній "*row*" і "*rowgutter*".

При визначенні стилю елементів, використовуючи ім'я grid-ліній і їх порядковий номер, ми можемо явно вказати за допомогою властивостей *grid-column* і *grid-row*, де саме повинен розташовуватися елемент:

grid-column: column 2; grid-row: row 1;

Це означає: елемент розташовується в першому рядку, який починається з грід-лінії *row*, і в стовпці, який починається з grid-лінії "*column*".

Отримуємо такий грід:

Рис. 3.85. Іменовані grid-лінії і функція repeat

3.8.11. Області гріду

У межах грід-контейнера можна визначати **грід-області (grid area)**. Грід-область визначається за допомогою двох вертикальних і двох горизонтальних grid-ліній, які і задають займаний нею простір. Область не еквівалентна одній комірці гріду і може включати кілька комірок. Області корисні для визначення семантичних відношень між різними частинами макету сторінки.

Для визначення областей у grid-контейнері застосовується **властивість *grid-template-areas***.

Наприклад, визначимо грід-контейнер та три області, які в ньому міститимуться:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width" />
  <title>Grid Layout в CSS3: області гріду </title>
  <style>
   *{ box-sizing: border-box; }
   html, body { margin:0px; padding:0px;}
   .grid-container {
      display: grid; height: 100vh;
      grid-template-columns: 150px  10px 1fr;
      grid-template-rows: 100px 10px 1fr 100px;
      grid-template-areas: "header header header"    ". . ."
            "sidebar . content"    "sidebar . content";
   }
   .header { grid-area: header;
              background-color: #98ABD3; }
   .sidebar { grid-area: sidebar;
              background-color:  #DCC0D9;}
   .content { grid-area: content;
              background-color: #E4E4E6; }
   .styletext { text-align: center; font-size: 1.8em;
                font-weight:bold; padding-top:25px; }
  </style>
 </head>
```

```
<body>
  <div class="grid-container">
    <div class="header styletext">область header</div>
    <div class="sidebar styletext">область sidebar</div>
    <div class="content styletext">область content</div>
  </div>
</body>
</html>
```

У grid-контейнері визначено три стовпці і чотири рядки, тому в підсумку в гріді матимемо 4х3=**12 комірок**. Але в розмітці сторінки визначено три елементи з однойменними областями: *header*, *sidebar*, *content*. І **властивість *grid-template-areas*** встановлює, як ці області будуть розташовуватися в комірках гріду:

- вираз "***header header header***" представляє перший рядок і вказує, що область *header* займає три комірки підряд;
- другий рядок подано виразом "**. . .**" , де кожна з крапок означає комірку, яка не належатиме жодній області і залишиться незаповненою; щоб залишити три клітинки незаповненими, вказуються три крапки, що розділяються пробілами; незаповнені комірки, в даному випадку, відіграють роль відступу;
- наступний вираз " ***sidebar . content*** " вказує на третій рядок, де область *sidebar* займає першу комірку, *content* – третю, а крапка відповідає порожній області, що є відступом між ними;
- четвертий рядок повторює третій.

У підсумку, схематичне розташування комірок із позначенням областей має вигляд:

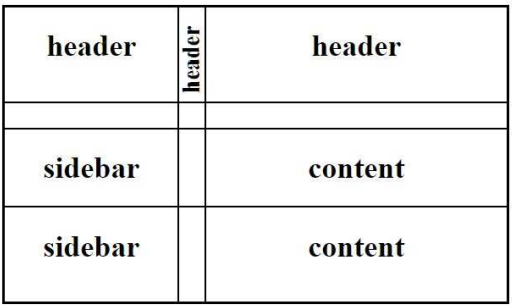


Рис. 3.86. Схема розмітки сторінки з використанням grid-областей

Для задання області в елементів використовується **властивість *grid-area***. Напр., елемент з класом *header* розміщується в область *header*.

Таким чином, сторінка має вигляд:

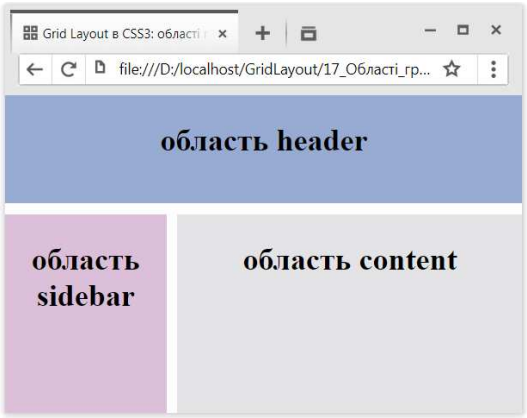


Рис. 3.87. Приклад використання grid-областей

3.8.12. Макет сторінки в Grid Layout

Розглянемо створення найпростішого **адаптивного стандартного макету веб-сторінки**, який складається з шапки, підвалу, основного вмісту, блоку навігації і сайдбару.

При створенні макету використаємо іменовані області, тому що досить зручно пов’язувати стилі із семантикою сторінки через області.

Визначимо наступну веб-сторінку:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width" />
    <title> Макет сторінки на Grid Layout </title>
    <style>
      *{ box-sizing: border-box; }
      html, body { margin:0; padding:0;}
      .grid-container {
```

```
      height:100vh;
      display: grid;
      grid-template-areas: "header"  "."  "menu"  "."
                "content"  "."  "sidebar"  "."  "footer";
      grid-template-columns: 1fr;
      grid-template-rows:
                  80px 8px 80px 8px 1fr 8px 80px 8px 80px;
    }
    .header { grid-area: header;
                  background-color: #98ABD3; }
    .menu { grid-area: menu;
                  background-color: #DCC0D9; }
    .sidebar { grid-area: sidebar;
                  background-color: #DCC0D9; }
    .content { grid-area: content;
                  background-color: #E4E4E6; }
    .footer { grid-area: footer;
                  background-color: #98ABD3; }
    .styletext { text-align: center; font-size: 1.8em;
              font-weight:bold; padding-top:25px; }
    @media screen and (min-width: 468px) {
    .grid-container {
      height:100vh;
      display: grid;
      grid-template-areas:
                  "header header header header header"
                  ". . . . ."
                  "menu . content . sidebar"
                  ". . . . ."
                  "footer footer footer footer footer";
      grid-template-columns: 130px 8px 1fr 8px 130px;
      grid-template-rows: 90px 8px 1fr 8px 90px;
     }
    }
  </style>
</head>
```

```
  <body>
    <div class="grid-container">
      <div class="header styletext">Header</div>
      <div class="content styletext">Content</div>
      <div class="menu styletext">Menu</div>
      <div class="sidebar styletext">Sidebar</div>
      <div class="footer styletext">Footer</div>
    </div>
  </body>
</html>
```

Оскільки передбачається можливість перегляду вебсторінки на мобільних пристроях, де ширина екрану обмежена, то хотілося б мати **адаптивний макет**. Тому наявні два різні визначення гріду.

Одне визначення гріду для мобільних пристроїв, умовно як ширину пристроїв вибрано значення в 468px. У такому випадку грід має тільки один стовпець і 5 рядків для кожної області плюс 4 рядки-роздільники:

Рис. 3.88а. Макет сторінки на Grid Layout для мобільних пристроїв

При збільшенні ширини екрану в дію вступає **інше визначення гріду**. Тоді стовпців і рядків буде по п’ять:

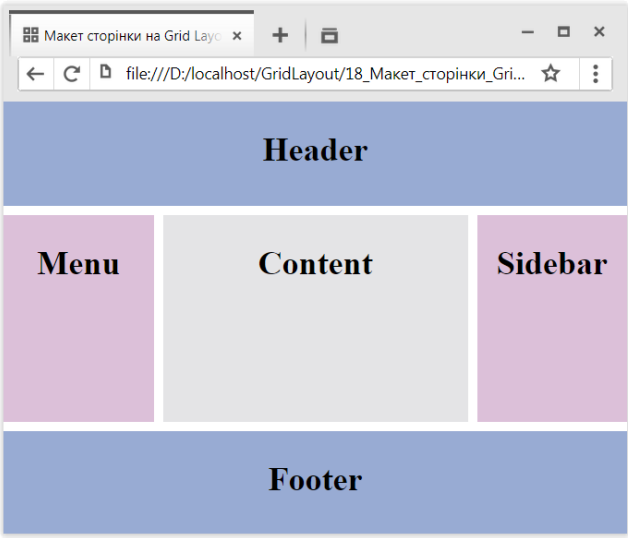


Рис. 3.88б. Макет сторінки на Grid Layout

4. ФРЕЙМВОРК BOOTSTRAP 4

4.1. Загальні відомості

Фреймворк Bootstrap – це безкоштовний набір інструментів із відкритим кодом, призначений для розробки веб-сайтів та веб-додатків, який містить шаблони HTML та CSS для форм, типографіки, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення мови програмування JavaScript. Він спрощує розробку динамічних веб-сайтів та веб-додатків. Інакше кажучи, Bootstrap **–** це клієнтський фреймворк, тобто інтерфейс для користувача, на відміну від коду серверної частини, який знаходиться на сервері.

Основні інструменти Bootstrap:

1.**Сітки** – заздалегідь задані розміри колонок, які можна одразу використовувати. Наприклад, ширина колонки *140 px* відноситься до класу *.span2* (*.col-md-2* в третій версії фреймворка), який можна використовувати в CSS-описі документа.

2.**Шаблони** – гумовий або фіксований шаблон документа.

3.**Типографіка** – опис шрифтів, тобто визначення деяких класів для шрифтів, таких як код, цитати і т. п.

4.**Медіа** дозволяє впорядкувати відео та зображення.

5.**Таблиці** – класи для оформлення таблиць.

6.**Форми** – класи для оформлення форм та деяких подій, які відбуваються з ними.

7.**Навігація** – класи для оформлення панелей, вкладок, переходів по сторінках, меню та панелі інструментів.

8.**Алерти** – оформлення діалогових вікон, зокрема підказок та спливаючих вікон.

Bootstrap 4 – це майже повністю переписаний Bootstrap 3.

Перелік найсуттєвіших змін:

1.Веб-шрифти за замовчуванням (Helvetica Neue, Helvetica, Arial) інтегровані в Bootstrap 4 та замінені набором вихідних шрифтів для оптимального відтворення текстового контенту на будь-якому пристрої з будь-якою операційною системою.

2.Перехід від використання LESS до SASS.

3.Не підтримується IE8, IE9 і iOS 6.

4.Додана підтримка Flexbox, і потім відключена підтримка non-flexbox.

5.Зміна основної одиниці виміру з *px* на *rem*.

6.Збільшено глобальний розмір шрифту з 14*px* до 16*px*.

7.Новий компонент «картка», що узагальнює панелі та інші компоненти.

8.Вилучений шрифт іконок Glyphicons.

9.Вилучені компоненти пейджера.

10. Переписані майже всі компоненти, плагіни jQuery і документація.

4.2. Початок роботи з Bootstrap 4

Підключення Bootstrap 4

Існує **два способи підключення Bootstrap 4**, який складається із файлу стилів та файлу js-скриптів, на власному веб-сайті:

1. Підключити Bootstrap 4 з CDN:

<link rel="stylesheet" href = "https://maxcdn.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css">

<script src ="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js">

</script>

2. Завантажити Bootstrap 4 з *getbootstrap.com* та підключити його з відповідної директорії вашого сайту.

Крім цього, потрібно підключити:

- **js-бібліотеку jQuery**:

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">

</script>

- **js-бібліотеку Popper**:

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/ umd/popper.min.js">

</script>

Ці бібліотеки потрібні для JavaScript-компонентів (таких як спливаючі, модальні підказки, ...). Однак, якщо є необхідність у використанні тільки CSS-частини Bootstrap, то в їх підключенні немає потреби.

Перевагою підключення з CDN є те, що багато користувачів уже завантажили Bootstrap 4 при відвідуванні іншого сайту. В результаті, він буде завантажений з кешу при відвідуванні вашого сайту, що приведе до більш швидкого завантаження. **Перевагами підключення завантажених файлів Bootstrap** є те, що з їх допомогою можна використовувати Bootstrap 4 на локальному сервері, оскільки іноді швидкість інтернету може бути низькою, а підключення з локальної аудиторії не потребує інтернету.

Використання для мобільних пристроїв

Bootstrap 4 також орієнтований для розробки сайтів під мобільні пристрої. Стилі мобільних пристроїв є частиною базової платформи. Для забезпечення правильної візуалізації та масштабування доторків потрібно додати тег ***<meta>*** всередині елемента <*head*>:

<*meta name="viewport"*
content="width=device-width, initial-scale=1">

де перераховані параметри визначають:

- "*width = device-width*" – значення, що встановлює для веб-браузера пристрою ширину сторінки рівною ширині екрану пристрою;
- "*initial-scale=1*" – початковий коєфіцієнт масштабування при першому завантаженні сторінки користувачем: значення *1.0* визначає масштаб *1:1*, тобто відсутність масштабування.

Контейнер в Bootstrap – це базовий елемент, який використовується в системі сіток.

Bootstrap 4 надає наступні **контейнери класів** (рис. 4.1):

- ***.container*** – клас створює контейнер з фіксованою шириною:

<*div class="container"*>
.container
</*div*>

- ***.container-fluid*** – клас створює контейнер з повною шириною, який охоплює всю ширину видового екрану (завжди 100% зони перегляду):

<*div class="container-fluid"*>
.container-fluid

* *

Таким чином, блок *div* з класом *container-fluid* завжди охоплюватиме все ширину сторінки, а з класом *container* – деяку фіксовану її частину:

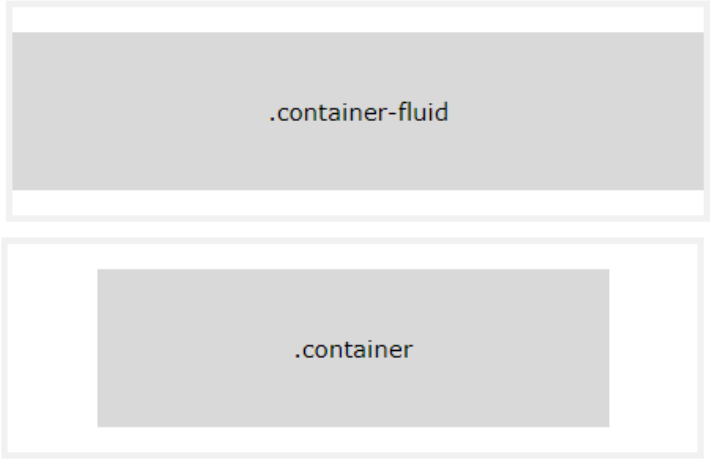


Рис. 4.1. Контейнери Bootsrap 4

4.3. Система сітки Bootstrap 4

Щоб зрозуміти, що таке **сітка в бутстрап**, сторінку розглядають як деяку таблицю, що складається з набору рядків. В кожному рядку рівно 12 стовпців, не більше і не менше. Тобто максимальна і мінімальна їх кількість – 12. Це означає, що кожен рядок сторінки поділений по ширині на 12 рівних частин. Рядків може бути скільки завгодно, оскільки сторінка може прокручуватися вниз скільки завгодно.

Що ми можемо робити із цими стовпцями? Ми можемо їх згрупувати, тобто об’єднувати в залежності від створюваного макету верстки (рис. 4.2).

Наприклад, сторінка сайту містить зліва блок меню, справа – блок основного вмісту. В цьому випадку ми можемо зробити так: три стовпці виділяємо під ліве меню і, відповідно, 9 стовпців – під основну частину. Таким чином, завдяки системі сіток ми створюємо розмітку сайту.

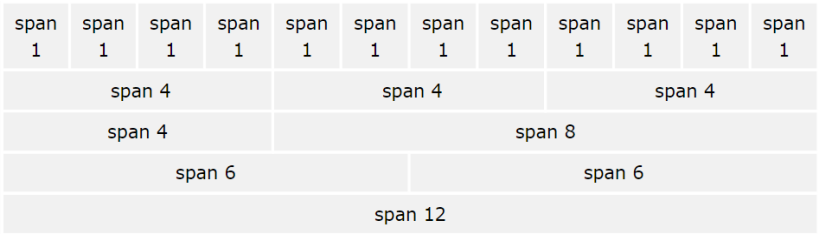


Рис. 4.2. Система сітки Bootstrap 4

Система сітки Bootstrap реагує на пристрій, на якому відображається сайт, тому стовпці перевпорядковуються **в залежності від розміру екрана**. Наприклад, на великому екрані сторінка сайту може бути організована в три колонки, а на маленькому – в дві чи одну.

Поведінка блоку на сторінці визначається одним або кількома класами *col*. **Синтаксис класу *col*:**

col-breakpoint-number_columns, де

breakpoint – це контрольна точка, яка визначає мінімальну ширину *viewport*, починаючи з якої клас буде діяти; в Bootstrap 4 – п’ять контрольних точок (без позначення, *sm*, *md*, *lg* та *xl*);

number_columns – ціле число від 1 до 12, яке визначає, скільки стовпців буде охоплювати елемент.

Таким чином, в Bootstrap 4 доступні для використання п’ять різновидів класу *col*:

- ***.col-*** (додаткові малі пристрої – ширина екрану менше 576 px);
- ***.col-sm-*** (малі пристрої – ширина екрану дорівнює або більше ніж 576 px);
- ***.col-md-*** (середні пристрої – ширина екрану дорівнює або перевищує 768 px);
- ***.col-lg-*** (*larger* великі пристрої – ширина екрану дорівнює або більше ніж 992 px);
- ***.col-xl-*** (*XLarge* пристрої – ширина екрану дорівнює або перевищує 1200 px).

Наведені класи можна комбінувати для створення гнучких та динамічних макетів. Кожен клас масштабується вгору, тобто якщо потрібно встановити однакову ширину *sm* і *md*, то

потрібно вказати тільки *sm*.

Наприклад, визначимо на сторінці два блоки:

```
<div class="row">
    <div class="col-sm-12 col-md-8 col-lg-6">col</div>
    <div class="col-sm-12 col-md-4 col-lg-6">col</div>
</div>
```

Ці два блоки на екрані телефону розташовуватимуться один під одним, розтягуючись на всю ширину екрану, на планшеті будуть нерівними, розташованими поряд, а на великому моніторі – однакові за розміром, навпіл, по ширині, ділитимуть сторінку.

Зверни увагу! Сума стовпців в сітці для кожного типу екрану (без позначення *sm*, *md*, *lg* та *xl*) повинна дорівнювати 12, інакше вони будуть нагромаджуватися.

Деякі правила системи сітки Bootstrap 4:

1. Рядки повинні бути розміщені в межах *.container-fluid* (повної ширини) або *.container* (фіксованої ширини) для коректного вирівнювання та заповнення.
2. Використання рядків для створення горизонтальних груп стовпців.
3. Контент розміщується всередині стовпців, та тільки стовпці можуть бути прямими нащадками рядків.
4. Стовпці створюють внутрішні лівий і правий відступи (*padding-left*, *padding-right*) між ними.
5. Стовпці сітки створюються методом задання 12 доступних стовпців, які необхідно охопити. Наприклад, три рівні стовпці можуть використовувати клас *.col-sm-4*.
6. Якщо ширина стовпців задана у відсотках, вони завжди є гнучкими і мають розмір відносно батьківського елемента.

Найсуттєвіша відмінність між третьою та четвертою версіями фреймворку полягає в тому, що четверта версія використовує **Flexbox**, а не властивість float. Однією з великих переваг Flexbox'у є те, що стовпці сітки без заданої ширини будуть автоматично розміщуватись як стовпці з однаковою шириною. Наприклад, три елементи з *.col-sm* будуть автоматично мати ширину 33,33%.

Flexbox не підтримується в IE9 і більш ранніх версіях. Якщо потрібна підтримка IE8-9, то можна використати Bootstrap 3. Ця

версія Bootstrap, як і раніше, підтримується командою для критичних виправлень і змін до документації. Тим не менше, до неї не додаються нові функції.

Розглянемо декілька прикладів:

1. Визначимо рядок, в який додамо потрібну кількість стовпців (тегів з класом *.col-*-**). Перша зірочка означає, що клас буде застосовуватися для всіх пристроїв з будь-якою розмірністю екрану (без позначення *sm*, *md*, *lg* або *xl*). А друга зірочка визначає кількість стовпців, які охоплює елемент.

```
<div class="row">
    <div class="col-*-*">col</div>
    <div class="col-*-*">col</div>
    <div class="col-*-*">col</div>
</div>
```

В цьому випадку Bootstrap, опрацювуючи макет, створює три колонки однакової ширини (33,33%), кожна з яких охоплює 4 стовпці даного рядка. Таким чином, друга зірочка дорівнює числу 4.

2. Розглянемо приклад, в якому Bootstrap, обробляючи макет, створює чотири однакові стовпці шириною 25%:

```
<div class="row">
    <div class="col">col</div>
    <div class="col">col</div>
    <div class="col">col</div>
    <div class="col">col</div>
</div>
```

Крім цього, стовпці однакової ширини на різних пристроях можна створити з використанням відповідного класу: *col-sm*, *col-md*, *col-lg*, *col-xl*.

Попередній приклад створює 4 однакові стовпці на всіх пристроях, а, наприклад, використання класу ***col-lg*** визначає стовпці однакової ширини на всіх пристроях, в яких ширина екрану дорівнює або більше *992 px*. При цьому на пристроях з шириною менше *992 px* стовпці будуть мати ширину 100% та розташуються один під одним.

Припустимо, що у нас є **простий макет з двома стовпцями**. При цьому стовпці повинні мати ширину 25% і 75% (рис. 4.3) та 33,33% і 66,66% (рис. 4.4) для всіх пристроїв.

Рис. 4.3. Поділ рядка на стовпці шириною 25% і 75%

Рис. 4.4. Поділ рядка на стовпці шириною 33,33% і 66,66%

У такому випадку HTML-структура матиме вигляд:

```
<!-- стовпці шириною 25% і 75% -->
<div class="container">
  <div class="row">
    <div class="col-3 bg-success"> <p>col-3</p> </div>
    <div class="col-9 bg-warning"> <p> col-9</p> </div>
  </div>
</div>
<!-- стовпці шириною 33.3% та 66.6% -->
<div class="container">
  <div class="row">
    <div class="col-4 bg-success"> <p> col-4</p> </div>
    <div class="col-8 bg-warning"> <p> col-8</p> </div>
  </div>
</div>
```

Bootstrap-класи *bg-success* та *bg-warning* будуть розглянуті в наступному параграфі.

Розглянемо **задачу відображення сторінки веб-сайту, яка поділена на дві вертикальні частини.** На пристроях з шириною екрану >= *768 px* ці частини повинні мати ширину 50% і 50% (рис. 4.5), а на пристроях з шириною < *768 px* та >= *576 px* – 25% і 75% (рис. 4.6).

HTML-структура сторінки матиме вигляд:

```
<div class="container">
  <div class="row">
    <div class="col-sm-3 col-md-6"> … </div>
    <div class="col-sm-9 col-md-6"> … </div>
  </div>
</div>
```

При цьому на додаткових малих пристроях (<*576 px*)

стовпці автоматично займатимуть 100% ширини сторінки (рис. 4.7).

Рис. 4.5. Відображення на пристроях із шириною >= *768 px*

Рис. 4.6. Відображення на пристроях із шириною >= *576 px* та < *768 px*

Рис. 4.7. Відображення на пристроях із шириною < *576 px*

4.4. Можливості Bootstrap 4

Крім побудови адаптивної сітки макетів веб-сторінок, Bootstrap 4 має набір можливостей. Розглянемо деякі з них:

1. Bootstrap 4 використовує значення, за замовчуванням, для ***font-size*** *16 px* та для ***line-height*** *1.5*. Крім того, всі елементи **<*p*>** мають *margin-top: 0* і *margin-bottom: 1rem* (*16 px*, за замовчуванням).

2. Стилі HTML-заголовків (**<*h1*>-<*h6*>**) визначені з більш оригінальним шрифтом та більшим розміром (рис. 4.8):

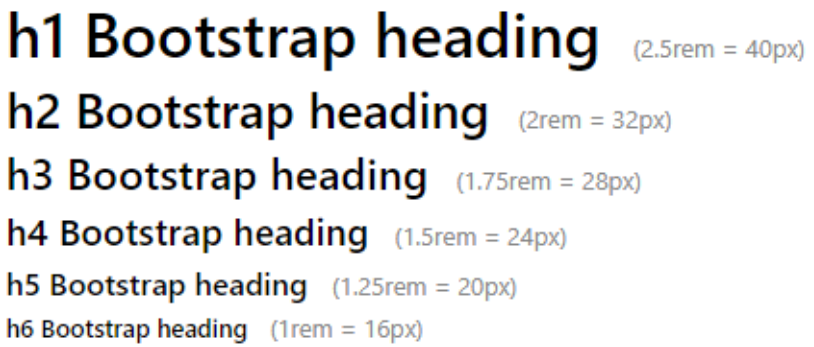


Рис. 4.8. Стилі заголовків у Bootstrap 4

3. Bootstrap 4 має деякі контекстні класи, які можуть бути використані для забезпечення "значення через кольори".

Класи, що задають колір тексту: *.text-muted*, *.text-primary*, *.text-success*, *.text-info*, *.text-warning*, *.text-danger*, *.text-secondary*, *.text-white*, *.text-dark*, *.text-body* (колір за замовчуванням) і *.text-light*.

Створимо сторінку, на якій проілюструємо використання вищеперерахованих класів (рис. 4.9):

```
<div class="container">
  <p class="text-muted">
     Це текст сірого кольору</p>
  <p class="text-primary">
     Це важливий текст (синій)</p>
  <p class="text-success">
     Цей текст вказує на успіх (зелений)</p>
  <p class="text-info">
     Цей текст показує деяку інформацію
     (світло-блакитний)</p>
  <p class="text-warning">
     Це текст – попередження (жовтий)</p>
  <p class="text-light">
     Це світло-сірий текст</p>
  <p class="text-white">
     Це білий текст</p>
  <p class="text-danger">
     Цей текст вказує на небезпеку (червоний)</p>
  <p class="text-secondary">
     Другорядний текст (сірий)</p>
  <p class="text-dark">
     Це темно-сірий текст </p>
  <p class="text-body">
     Колір за замовчуванням</p>
</div>
```

Візуально, наповнення сторінки має вигляд (рис. 4.9), при цьому “*Це світло-сірий текст*”, “*Це білий текст*” виділені курсором миші для того, щоб їх можна було помітити.

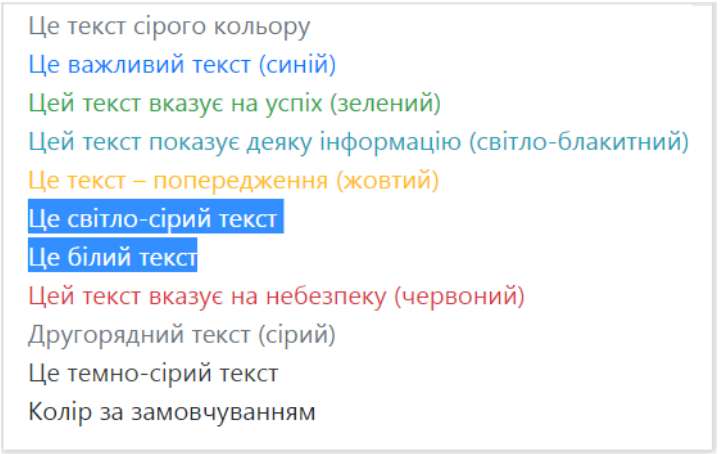


Рис. 4.9. Використання класів Bootstrap 4 для задання кольору тексту

Для задання кольору фону використовуються наступні класи: *.bg-primary*, *.bg-success*, *.bg-info*, *.bg-warning*, *.bg-danger*, *.bg-secondary*, *.bg-dark* і *.bg-light* (рис. 4.10). Кольори фону не задають колір тексту, тому можна використовувати їх разом з *.text-** класом.

Рис. 4.10. Використання класів Bootstrap 4 для задання кольору фону

4. Базовий стиль таблиці *.table* у Bootstrap 4 має горизонтальні розділювачі світлого кольору.

Клас *.table* можна застосувати до будь-якої таблиці:

<table class="table"> …
</table>

При додаванні класів *.table-dark (чорний фон таблиці), .table-striped (додає до таблиці зебра-смуги):*

<table class="table-dark table-striped">…
</table>

буде створена таблиця, стилізована наступним чином (рис. 4.11):

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Рис. 4.11. Стилі Bootstrap 4 для таблиць

5. При виведенні зображень на сторінку використовують класи Bootstrap для зміни стилю виводу (рис. 4.12):

.rounded виконує заокруглення кутів зображення:
<img src="town.jpg" class="rounded" alt="Old town">
.rounded-circle виводить зображення у вигляді круга:
<img src="city.jpg" class="rounded-circle" alt="New town ">
.img-thumbnail *перетворює зображення на мініатюру:*
<img src="Bridge.jpg" class="img-thumbnail" alt=" Bridge">

а) *б)* *в)*

Рис. 4.12. Зображення на сторінці із використанням класів Bootstrap 4:
а – заокруглені кути; б – круг; в – мініатюра.

6. Bootstrap 4 надає простий спосіб **створення сповіщень**. Сповіщення створюються за допомогою **класу *.alert***, після якого вказується один із контекстних класів: *.alert-success*, *.alert-info*, *.alert-warning*, *.alert-danger*, *.alert-primary*, *.alert-secondary*, *.alert-light* або *.alert-dark*.

Наприклад, створимо сповіщення - застереження:

```
<div class="alert alert-warning">
  <strong>Увага!</strong>
    Попередження ...
</div>
```

Сповіщення, створене *HTML*-кодом вище, має вигляд:

Рис. 4.13. Сповіщення - застереження

7. Аналогічно як для сповіщень, можна визначити класом Bootstrap і **стилі для кнопок**.

Для цього потрібно застосувати **клас *.btn*** та деякий відповідний клас для елемента *<button>*. Цим відповідним класом може бути: *.btn-primary*, *.btn-secondary*, *.btn-success*, *.btn-info*, *.btn-warning*, *.btn-danger*, *.btn-dark*, *.btn-light*, *.btn-link* (рис. 4.14). Наприклад:

```
<button type="button" class="btn">Basic</button>
<button type="button" class="btn btn-primary">
        Primary</button>
…
```

Рис. 4.14. Кнопки зі стилями Bootstrap 4

Крім цього, з використанням класів фреймворку можна задати **контурні кнопки** (*.btn-outline-primary*, *.btn-outline-secondary*, ...), **величину кнопок** (*.btn-lg*, *.btn-sm*), **кнопки рівня блоку**, які будуть охоплювати всю ширину батьківського елемента (*.btn-block*), неактивні кнопки (*.disabled*).

8. За допомогою класів Bootstrap 4 можна стилізувати **групи кнопок**.

Клас *.btn-group* дозволяє згрупувати набір кнопок разом в один рядок:

```
<div class="btn-group">
  <button type="button"
          class="btn btn-primary">Apple</button>
  <button type="button"
          class="btn btn-primary">Lenovo</button>
</div>
```

Аналогічно, **клас *.btn-group-vertical*** створює вертикальну групу кнопок.

Наприклад, згрупуємо кнопки в один ряд та створимо з останньої з них випадаюче меню (детальніше, описано в пункті 10 цього підрозділу):

```
<div class="btn-group">
  <!-- Звичайні кнопки -->
  <button type="button"
          class="btn btn-primary">Apple</button>
  <button type="button"
         class="btn btn-primary">Lenovo</button>
  <!-- Кнопка з випадающим меню -->
  <div class="btn-group">
    <button type="button"
          class="btn btn-primary dropdown-toggle"
          data-toggle="dropdown"> Samsung
    </button>
    <div class="dropdown-menu">
      <a class="dropdown-item" href="#"> Планшети</a>
      <a class="dropdown-item" href="#">Смартфони</a>
    </div>
  </div>
</div>
```

На сторінці створені кнопки виглядають наступним чином:

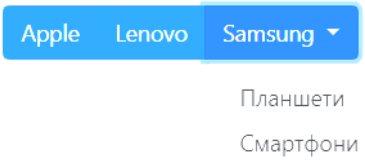


Рис. 4.15. Група кнопок, створена класами Bootstrap 4

9. Якщо веб-сайт має велику кількість сторінок, то доцільно додати певну **пагінацію** (розбивку на сторінки) на кожній сторінці (рис. 4.16).

Щоб створити основну розбивку на сторінки, потрібно додати клас *.pagination* до **елемента *<ul>***. Далі, додати до **кожного елемента *<li>*** клас *.page-item*, а **для кожного посилання** всередині *<li>* – клас *.page-link*.

Клас *.active* використовується для виділення поточної сторінки, клас *.disabled* – для посилань, які не є активними.

Розглянемо приклад пагінації:

```
<ul class="pagination">
   <li class="page-item">
      <a class="page-link" href="#">Previous</a>
   </li>
   <li class="page-item disabled">
      <a class="page-link" href="#">1</a>
   </li>
   <li class="page-item">
      <a class="page-link" href="#">2</a>
   </li>
   <li class="page-item active ">
   <a class="page-link" href="#">3</a></li>
   <li class="page-item">
      <a class="page-link" href="#">Next</a>
   </li>
</ul>
```

Створена пагінація має такий вигляд:

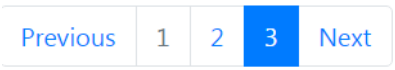


Рис. 4.16. Пагінація з використанням класів Bootstrap 4

10. Випадаюче меню являє собою меню, що розкривається при натисканні на кнопку або посилання (рис. 4.17). Воно дозволяє користувачеві вибрати одне зі значень списку. Для створення такого меню потрібно:

1) спочатку задати "**обгортку**" – елемент *div* **із класом *.dropdown***; клас *.dropdown* використовується для задання елементу відносного позиціонування (*position: relative*);

2) додати **кнопку (або посилання) для розкриття меню**, тобто елемент з класом *.dropdown-toggle* та атрибутом *data-toggle="dropdown"*;

3) створити **блок меню** – елемент *div* з класом *.dropdown-menu*, що міститиме набір елементів меню. І до кожного елемента меню (посилання чи кнопки) додати **клас *.dropdown-item***.

Проілюструємо на прикладі (рис. 4.17):

```
<div class="dropdown">
  <button type="button"
        class="btn btn-primary dropdown-toggle"
        data-toggle="dropdown"> Випадаюче меню
  </button>
  <div class="dropdown-menu">
     <a class="dropdown-item" href="#">
        Посилання 1</a>
     <a class="dropdown-item" href="#">
        Посилання 2</a>
     <a class="dropdown-item" href="#">
        Посилання 3</a>
     <div class="dropdown-divider"></div>
     <a class="dropdown-item" href="#">
        Інше Посилання</a>
  </div>
</div>
```

Клас *.dropdown-divider* використовується для створення розділювача між посиланнями у вигляді тонкої горизонтальною лінії:

```
<div class="dropdown-divider"></div>
```

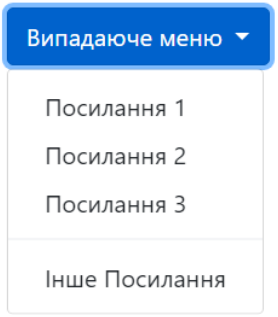


Рис. 4.17. Випадаюче меню в Bootstrap 4

11. Крім наведених вище можливостей, даний фреймворк має стилі та скрипти для

- стилізації карток (англ. *cards*), форм, списків, навігації;
- створення індикатора виконання, каруселів, модальних вікон, спливаючих підказок;
- виконання пошуку в таблиці.

4.5. Висновки

Bootstrap – це відкритий і безкоштовний HTML, CSS і JS фреймворк, який використовується веб-розробниками для швидкого створення адаптивних дизайнів веб-сайтів. Фреймворк Bootstrap використовується не лише незалежними розробниками, а й великими компаніями. Основна сфера його застосування – це розробка фронтенд складових сайтів та інтерфейсів адмінок. Серед аналогічних систем фреймворк Bootstrap найпопулярніший.

Фактично, Bootstrap – це набір CSS- та JavaScript-файлів, після підключення яких буде доступна для верстки велика кількість класів та готових компонентів. Використовуючи їх, можна швидко та якісно розробити сучасний адаптивний дизайн веб-сайту.

Класи фреймворку Bootstrap можна розділити на 3 великі групи:

1.Класи створення сітки (адаптивного макета сторінки).

2.Класи стилізації контенту (тексту, коду, зображень, таблиць та іншої інформації).

3.Службові класи (для вирішення найбільш поширених допоміжних задач, таких як вирівнювання, управління відображенням та ін.).

Крім класів, фреймворк Bootstrap має **готові об'єкти**, зокрема кнопки, хлібні крихти, форми, навігаційні меню, випадаючі списки, спливаючі підказки та ін.

Застосування фреймворка Bootstrap при створенні сайтів забезпечує швидку розробку якісних сучасних адаптивних дизайнів сайтів. Для роботи із даним фреймворком не потрібні глибокі знання з HTML, CSS, JavaScript та jQuery. Крім цього, він є кросбраузерним і кросплатформним, оскільки адаптований під усі популярні операційні системи і браузери, а також є відкритим і безкоштовним.

Але, крім переваг, Bootstrap має також **недоліки**. **Перший** в тому, що він не підходить для проектів з унікальним дизайном. В цьому випадку, з Bootstrap для проекту використовують в основному тільки сітку. **Другий** полягає в тому, що для проекту потрібні не всі компоненти і класи Bootstrap, а тільки деякі. Але цей недолік не є проблемою, оскільки можна створити свою збірку, що складається тільки з потрібних класів та компонентів Bootstrap.

5. МОВА ПРОГРАМУВАННЯ JAVASCRIPT

5.1. Основи JavaScript

5.1.1. Загальні відомості

На сьогодні світ веб-сайтів складно уявити без **мови програмування JavaScript**. JavaScript – це те, що робить живими веб-сторінки. Спочатку мова JavaScript мала досить невеликі можливості. Її мета полягала лише в тому, щоб доповнити **веб-сторінку певною поведінкою**. Однак розвиток веб-середовища, поява HTML5 і технології Node.js відкрили перед JavaScript набагато більші горизонти. Тепер JavaScript продовжує використовуватися для створення веб-сайтів, але надає набагато більше можливостей, зокрема, застосовується **як мова серверної сторони**. Тобто, якщо раніше JavaScript застосовувалася тільки на веб-сторінці, а на стороні сервера потрібно було використовувати такі технології, як PHP, ASP.NET, Ruby, Java, то зараз завдяки Node.js ми можемо **обробляти всі запити до сервера, використовуючи JavaScript**.

Останнім часом активно розвивається сфера мобільної розробки. І JavaScript знову ж таки не залишається осторонь: збільшення потужності пристроїв і повсюдне поширення стандарту HTML5 привело до того, що **для створення мобільних додатків** можна використовувати JavaScript. Більше того, завдяки виходу нового сімейства операційних систем Windows: **Windows 8/8.1/10** – можна використовувати цю мову програмування **для розробки додатків для них**. Тобто JavaScript вже переступила межі веб-браузера, які були окреслені при її створенні. Крім цього, JavaScript може **використовуватися в галузі розробки IoT** (Internet of Things або інтернет-речей) для програмування найрізноманітніших “розумних” пристроїв, які взаємодіють з інтернетом. Таким чином, можна зустріти застосування JavaScript практично всюди. Це дійсно одна з найпопулярніших мов програмування, і її поширеність продовжуватиме зростати.

JavaScript є мовою, що інтерпретується. Це означає, що код на мові JavaScript виконується за допомогою

інтерпретатора. Інтерпретатор отримує оператори JavaScript, які визначені в коді веб-сторінки, виконує їх (або інтерпретує). **Для розробки на JavaScript потрібно** текстовий редактор для написання коду і веб-браузер для його тестування. Як текстовий редактор можна використовувати Notepad++. Також існують різні середовища розробки, які підтримують JavaScript і полегшують розробку, наприклад Visual Studio, WebStorm, Netbeans та ін. При бажанні їх можна також використовувати.

5.1.2. Перша програма на JavaScript

Створимо першу програму на JavaScript. Спочатку визначимо для нашого додатку каталог його розташування, наприклад, **папка з назвою *app***, у якій створимо файл ***index.html***. Цей файл являє собою веб-сторінку. Відкриємо його в текстовому редакторі, наприклад в Notepad++, і визначимо в файлі наступний код:

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8" />
    <title>Мова JavaScript</title>
    <script> alert("Секція head"); </script>
 </head>
 <body>
   <h2>Перший заголовок</h2>
   <script> alert("Перший заголовок"); </script>
   <h2>Другий заголовок</h2>
   <script> alert("Другий заголовок"); </script>
 </body>
</html>
```

Тут визначені стандартні елементи *html*. **В елементі *head*** визначається кодування utf-8 і заголовок (елемент *title*). **В елементі *body*** визначається тіло веб-сторінки, що складається з елементів *<h2>* і *<script>*.

Підключення коду javascript на html-сторінці здійснюється за допомогою **тега *<script>***. Цей тег слід розміщувати або в заголовку (між тегами *<head>* і *</head>*), або в тілі веб-сторінці (між тегами *<body>* і *</body>*). Часто

підключення скриптів відбувається перед закриваючим тегом *</body>* для оптимізації завантаження веб-сторінки.

Раніше потрібно було в тезі *<script>* вказувати тип скрипта, оскільки цей тег може використовуватися не тільки для підключення коду javascript, але і для інших цілей. Так, навіть зараз можна зустріти на деяких веб-сторінках таке визначення елемента *script*:

<script type = "text / javascript">

Але в наш час **атрибут *type*** переважно опускається, тому що браузери, за замовчуванням, вважають, що елемент *script* містить оператори javascript. Код javascript у наведеному вище прикладі містить оператори тільки одного типу:

alert('…');

Код javascript може містити багато операторів і кожний оператор завершується крапкою з комою. Наш оператор викликає метод *alert()*, який виводить діаголове вікно із заданим, у лапках, повідомленням.

При роботі в текстовому редакторі з використанням кириличних символів потрібно враховувати кодування. На нашій веб-сторінці використовувалося кодування utf-8. Також потрібно задати це кодування для веб-сторінки і в текстовому редакторі. Для цього в меню кодувань вибираємо пункт "***Перетворити в UTF-8 без BOM***".

Відносно виконання коду javascript: браузер отримує веб-сторінку з кодом html і javascript, після чого він її інтерпретує. Результат інтерпретації у вигляді різних елементів – кнопок, полів вводу, текстових блоків і т.д., можна побачити в браузері. Інтерпретація веб-сторінки відбувається послідовно зверху вниз.

Коли браузер зустрічає на веб-сторінці елемент *<script>* з кодом javascript, вступає в дію **вбудований інтерпретатор javascript**. І поки він не завершить свою роботу, далі інтерпретація веб-сторінки не йде.

У наведеному прикладі є три вставки коду javascript – один в секції *<head>* і по одному після кожного заголовка. **При відкритті веб-сторінки** в браузері, незважаючи на те, що в коді оголошено два заголовки, в браузері їх ще не видно, оскільки спочатку виконується код javascript з секції *head*, в результаті чого на екран буде виведено **діалогове вікно з повідомленням**

"Секція head". Поки воно не буде закрито, інтерпретація веб-сторінки не продовжиться. Після закриття вікна з повідомленням браузер додасть на сторінку перший заголовок і знову зупиниться на наступному блоці коду javascript. Результат виконання якого – **діалогове вікно з повідомленням *"Перший заголовок"***, після закриття якого браузер продовжить інтерпретацію і додасть на сторінку другий заголовок, зупиниться на третьому блоці коду javascript.

Після закриття **діалогового вікна з повідомленням *"Другий заголовок"*** браузер завершить інтерпретацію, і веб-сторінка буде повністю завантажена. Цей момент дуже важливий, оскільки може впливати на продуктивність. Тому часто вставки коду javascript розміщують перед закриваючим тегом *</body>*, коли основна частина веб-сторінки вже завантажена в браузер.

Ще один спосіб підключення коду JavaScript на веб-сторінку представляє винесення коду в зовнішні файли і їх підключення за допомогою **тега *<script>***, в якому задається **атрибут *src***. Цей атрибут вказує на шлях до файлу скрипта. Можливий випадок використання відносного та абсолютного шляху. Якщо веб-сторінка знаходиться в одній папці з каталогом js, то можна написати відносний шлях, наприклад js/myscript.js. Також треба враховувати, що за відкриваючим тегом *<script>* повинен обов'язково розміщуватися закриваючий тег *</script>*:

<script src = "js/myscript.js"> </ script>

На відміну від визначення коду javascript всередині html-файлу, **винесення його в зовнішні файли має ряд переваг**:

- повторне використання одного і того ж коду на кількох веб-сторінках;
- можливість кешування зовнішніх файлів, за рахунок чого при наступному зверненні до сторінки браузер знижує навантаження на сервер, а браузеру треба завантажити менший обсяг інформації;
- більш «чистий» код веб-сторінки, оскільки він містить тільки html-розмітку, а код поведінки зберігається в зовнішніх файлах.

Тому рекомендується використовувати код javascript в зовнішніх файлах, а не прямі вставки на веб-сторінку за допомогою елемента *script*.

Незамінний інструмент при роботі з JavaScript – **консоль браузера**, яка дозволяє проводити налагодження програми. У багатьох сучасних браузерах є подібна консоль. Наприклад, щоб відкрити консоль у Google Chrome, потрібно перейти в меню *«Інші інструменти» → «Інструменти розробника»*.

Для виведення різного роду інформації в консоль браузера використовується **спеціальна функція *console.log()***. Наприклад, визначимо такий код:

```
var a = 5 + 8;
console.log("Результат операції ");
console.log(a);
```

У коді вище за допомогою **ключового слова *var*** оголошується змінна *a*, якій присвоюється сума двох чисел 5 і 8: *var a = 5 + 8*. За допомогою **методу *console.log()*** виводиться значення змінної *a*. Після запуску веб-сторінки в браузері в консолі можна побачити результат виконання коду.

Також може бути корисний **метод *document.write()***, який виводить інформацію на веб-сторінку. Наприклад, візьмемо попередній приклад і змінимо в ньому метод *console.log* на *document.write*:

```
var a = 5 + 8;
document.write( "Результат операції ");
document.write(a);
```

Програма виводить результат операції безпосередньо на веб-сторінку.

5.1.3. Змінні, константи та типи даних

Для зберігання даних в програмі використовуються **змінні**. Змінні призначені для зберігання будь-яких тимчасових даних або таких, які в процесі роботи можуть змінювати своє значення. Для створення змінних застосовуються **ключові слова *var* і *let***. Наприклад, оголосимо змінну *myIncome*:

```
var myIncome;
let myIncome2; // інший варіант
```

Кожна змінна має ім'я. Ім'я являє собою набір алфавітно-цифрових символів, знака підкреслення (_) або знака долара ($), причому ім'я змінної не може починатися з цифрових символів. Тобто в назві змінної можна використовувати літери, цифри, знак підкреслення та знак долара, однак всі інші символи заборонені. Наприклад, правильні назви змінних:

- *$commision;*
- *someVariable;*
- *product_Store;*
- *income2;*
- *myIncome_from_deposit*.

Наступні назви некоректні і не можуть використовуватися:

- *222lol;*
- *@someVariable;*
- *my%percent.*

Також не можна давати змінним такі імена, які збігаються з зарезервованими ключовими словами. В JavaScript не так багато ключових слів, тому цього правила не складно дотримуватися. Наприклад, наступне визначення змінної є некоректним, оскільки *for* – ключове слово в JavaScript:

var for;

Перелік зарезервованих слів у JavaScript: *abstract, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, enum, export, extends, false, final, finally, float, for, function, goto, if, implements, import, in, instanceof, int, inteface, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, volatile, void, while, with.*

Потрібно враховувати, що **мова JavaScript є чутливою до регістру**, тому в наступному коді оголошено дві різні змінні:

var myIncome;
var MyIncome;

Через кому можна визначати одразу кілька змінних:

var myIncome, procent, sum;
let a, b, c;

За допомогою знака «=» (який ще називають оператором присвоєння) можна **присвоювати змінній значення**:

var income = 300;

let price = 76;

Процес присвоєння змінній початкового значення називається **ініціалізацією**. Тепер змінна *income* буде зберігати число 300, а змінна *price* – число 76. Також можна змінити значення змінних:

var income = 300;

income = 400;

console.log(income);

За допомогою **ключового слова *const*** можна визначити **константу**, яка, як і змінна, зберігає значення, проте це значення не може бути змінено:

const rate = 10;

При спробі змінити значення константи виникне помилка:

const rate = 10;

rate = 23; // помилка, rate – константа

Оскільки не можна змінити значення константи, то вона повинна бути ініціалізована при оголошенні. Якщо цього не зробити, виникне помилка:

const rate; // помилка, rate НЕ ініціалізована

Всі використовувані дані в JavaScript мають певний тип. Всього в JavaScript **п'ять примітивних типів даних**:

- ***string*** представляє рядок;
- ***number*** представляє числове значення;
- ***boolean*** представляє логічне значення *true* або *false*;
- ***undefined*** вказує, що значення не встановлено (поки змінна не отримує свого початкового значення, має значення *undefined*); значення *undefined* формує окремий тип *undefined* із одного значення – самого себе;
- ***null*** – окремий тип, що містить тільки одне значення *null* (аналогічно як і *undefined*), що означає невизначене значення.

Усі дані, які не потрапляють до перерахованих вище п'яти типів, відносяться до **типу object**.

Числа в JavaScript можуть мати дві форми: цілі числа з діапазону від -2^{53} до 2^{53}; дробові числа (числа з плаваючою

точкою) з того ж діапазону. Як роздільник цілої і дробової частин, як і в інших мовах програмування, використовується крапка.

Тип string представляє рядки, тобто дані, які вкладені в лапки, наприклад *"Привіт світ"*. Причому можна використовувати як подвійні, так і одинарні лапки. Єдине обмеження: тип закриваючих лапок повинен бути той самий, що й тип відкриваючих, тобто або обидві подвійні, або обидві одинарні.

Якщо всередині рядка зустрічаються лапки, то їх потрібно екранувати слешем. Наприклад, нехай є текст *"Бюро "Рога і копита" "*. Тепер, екрануючи лапки:

var companyName = "Бюро \" Рога і копита \"";

Також можна всередині рядка використовувати інший тип лапок:

var companyName1 = "Бюро 'Роги і копита'";

var companyName2 = 'Бюро "Рога і копита"';

Тип boolean представляє булеві або логічні значення *true* і *false* (тобто так чи ні):

var isAlive = true;

var isDead = false;

Часто виникає плутанина **між *null* і *undefined***. Отже, якщо визначено змінну без присвоєння початкового значення, то вона має тип *undefined*:

var isAlive;

console.log(isAlive); // виведе undefined

Присвоєння значення *null* означає, що змінна має деяке невизначене значення (не число, не рядок, не логічне значення), але все-таки має значення (*undefined* означає, що змінна не має значення):

var isAlive;

console.log(isAlive); // виведе undefined

isAlive = null;

console.log(isAlive); // виведе null

isAlive = undefined; // знову встановимо тип undefined

console.log(isAlive); // undefined

Тип object представляє складений об’єкт. Найпростіше визначення об’єкта через фігурні дужки:

var user = {};

Об’єкт може мати різні властивості і методи:

var user = {name: "Tom", age: 24};

console.log (user.name); //Tom

У цьому випадку об’єкт називається *user*, він має дві властивості: *name* і *age*. Це короткий опис об’єктів, більш докладний опис наводиться у відповідній главі.

JavaScript є мовою зі **слабкою типізацією**. Це означає, що змінні можуть динамічно змінювати тип. Наприклад:

var X; // mun undefined

console.log(X);

X = 45; // mun number

console.log(X);

X = "45"; // mun string

console.log(X);

У другому і третьому випадках консоль виведе число 45, але в другому випадку змінна *X* містить число, а в третьому – рядок. Це важливий момент, який потрібно враховувати і від якого залежить поведінка змінної в програмі:

var X = 45; // mun number

var Y = X + 5;

console.log(Y); // 50

X = "45"; // mun string

var Z = X + 5

console.log(Z); // 455

Вище в обох випадках до змінної *X* застосовується операція додавання (+). Але в першому випадку, *X* містить число, тому результатом операції *X + 5* буде число 50.

У другому випадку *X* містить рядок. Але операція додавання між рядком і числом 5 неможлива. Тому число 5 перетворюватиметься до рядка і буде відбуватися операція об’єднання рядків. Результатом виразу *X + 5* буде рядок “455”.

За допомогою **оператора *typeof*** можна отримати тип змінної:

var name = "Yuzik";

console.log(typeof name); // string

var x = 45.8;

console.log(typeof x); // number

```
var flag = true;
console.log(typeof flag); // boolean
var y;
console.log(typeof y); // undefined
```

5.1.4. Операції

JavaScript підтримує всі **базові математичні операції**: додавання, віднімання, множення, ділення, ділення по модулю (оператор %) – операція, яка повертає цілочислову остачу від ділення, інкремент (збільшення на одиницю) та декремент (зменшення на одиницю).

Існує **префіксний інкремент/декремент**, який спочатку збільшує/зменшує змінну на одиницю і потім повертає її значення. І є **постфіксний інкремент/декремент**, який спочатку повертає значення змінної, а потім збільшує/зменшує його на одиницю:

```
// префіксний інкремент
var n = 5;
var m = ++n;
console.log(n); // 6
console.log(m); // 6
// постфіксний інкремент
var k = 5;
var l = k++;
console.log(k); // 6
console.log(l); // 5
```

Як прийнято в математиці, всі операції виконуються зліва направо **залежно від їх пріоритету**: спочатку виконуються операції інкремента і декремента, потім – множення і ділення, після – додавання і віднімання. Щоб змінити стандартний хід виконання операцій, використовують дужки.

Операції присвоєння:

- = – оператор присвоює змінній вказане значення: `var x = 5;`
- += – додавання з подальшим присвоєнням результату, наприклад:

```
var a = 23;
```

a += 5; // аналогічно a = a + 5;

• -= – віднімання з наступним присвоєнням результату, наприклад:

var a = 28;

a -= 10; // аналогічно a = a - 10

• *= – множення з подальшим присвоєнням результату:

var x = 20;

*x *= 2; // аналогічно x = x * 2*

• /= – ділення з подальшим присвоєнням результату:

var x = 40;

x /= 4; // аналогічно x = x / 4

• %= – отримання остачі від ділення з подальшим присвоєнням результату:

var x = 10;

x%= 3; // аналогічно x = x%3

Для перевірки умови використовуються **оператори порівняння**. Оператори порівняння порівнюють два значення і повертають значення *true* або *false*:

• == – оператор рівності порівнює два значення, якщо вони рівні, повертає *true*, інакше повертає *false*: *x==5;*

• === – оператор тотожності порівнює два значення і їх тип, якщо вони рівні, повертає *true*, інакше повертає *false*: *x===5;*

• != – оператор порівняння двох значень, якщо вони різні, повертає *true*, інакше повертає *false*: *x!=5;*

• !== – оператор порівняння двох значень та їх типів, якщо вони різні, повертає *true*, інакше повертає *false*: *x!==5;*

• > – оператор порівняння двох значень, якщо перше більше другого, то повертає *true*, інакше повертає *false*: *x>5;*

• < – оператор порівняння двох значень, якщо перше менше другого, то повертає *true*, інакше повертає *false*: *x<5;*

• >= – оператор порівняння двох значень, якщо перше більше або дорівнює другому, то повертає *true*, інакше повертає *false*: *x>=5;*

• <= – оператор порівняння двох значень, якщо перше менше або дорівнює другому, то повертає *true*, інакше повертає *false*: *x<=5*.

Всі оператори досить прості, за винятком оператора рівності й оператора тотожності. Обидва порівнюють два значення, але оператор тотожності також враховує і тип значення. Наприклад:

```
var x = 100;
var y = "100";
var result = x == y;
console.log(result); // true
```

У результаті, змінна *result* дорівнює *true*, тому що фактично і *x*, і *y* зберігають число 100. Але оператор тотожності поверне в цьому випадку *false*, оскільки дані різного типу. Аналогічно працюють оператори нерівності != та !==.

Логічні операції застосовуються для об'єднання результатів двох операцій порівняння. В JavaScript є такі логічні операції:

- && – повертає *true*, якщо обидві операції порівняння повертають *true*, інакше повертає *false*;
- || – повертає *true*, якщо хоча б одна операція порівняння повертає *true*, інакше повертає *false*;
- ! – повертає *true*, якщо операція порівняння повертає *false*, і навпаки.

Можна використовувати **оператор + для об'єднання рядків**. Наприклад:

```
var name = "Тарас";
var surname = "Шевченко"
var fullname = name + " " + surname;
console.log(fullname); //Тарас Шевченко
```

Якщо один із виразів рядок, а інший – число, то число буде перетворено до рядка і виконається операція об'єднання рядків:

```
var name = "Джордж";
var fullname = name + 111;
console.log(fullname); // Джордж111
```

5.1.5. Перетворення даних

Часто виникає необхідність перетворити дані з одного типу в інший. Наприклад:

```
var x = "55";
var y = "5";
```

var result = x + y;
console.log(result); // 555

Обидві змінні представляють рядки, а точніше рядкові представлення чисел. В результаті буде не число 60, а рядок 555. Було б непогано, якби їх теж можна було б додавати, віднімати, працювати як зі звичайними числами. В цьому випадку, можна використовувати операції перетворення. Для **перетворення рядка в число** застосовується функція *parseInt()*:

var x = "55";
var y = "5";
var result = parseInt(x) + parseInt(y);
console.log(result); // 60

Для **перетворення рядка в дійсне число** застосовується функція *parseFloat()*:

var x = "55.35";
var y = "4.65";
var result = parseFloat(x) + parseFloat(y);
console.log(result); //60

При цьому рядок може мати змішаний вміст, наприклад, *"123hello"*, тобто містити не тільки цифри, але є і звичайні символи. Метод *parseInt()*, в будь-якому випадку, спробує виконати перетворення:

var x = "123hello";
var y = parseInt(x);
console.log(y); // 123

Якщо метод не може виконати перетворення, то він повертає **значення NaN (Not a Number)**, яке означає, що рядок не являє собою число і не може бути перетворений в число. За допомогою **спеціальної функції *isNaN()*** можна перевірити, чи представляє рядок число. Якщо рядок не є числом, то функція повертає *true*, якщо це число, то *false*:

var x = "JavaScript";
var y = "33";
var result = isNaN(x);
console.log(result); // true – x не є числом
result = isNaN(y);
console.log(result); // false – y є числом

Вище розглянуто **перетворення рядка в число у десятковій системі числення (с/ч)**. При цьому інтерпретатор вважає, за замовчуванням, що рядок містить число в десятковій с/ч.

Однак **можна перетворити рядок із числом у довільній с/ч в число в десятковій системі**. При перетворенні інтерпретатор JavaScript автоматично визначає систему числення числа в рядку (як правило, вибирається десяткова система). Але, за допомогою другого параметру, можна явно вказати, що потрібно перетворити рядок з числом у довільній с/ч. Наприклад, перетворення рядка, що містить число в двійковій системі:

```
var x = "111";
var y = parseInt(x, 2);
console.log(y); // 7
```

Результат – число 7, оскільки 111 в двійковій системі – це число 7 в десятковій.

Напишемо невеликий код, в якому використаємо операції зі змінними:

```
var Sum = prompt("Введіть суму вкладу", 1000);
var Percent = prompt("Введіть процентну ставку", 10);
var Nsum = parseInt(Sum);
var Nprocent = parseInt(Percent);
Nsum = Nsum + Nsum * Nprocent / 100;
alert("Після нарахування відсотків сума вкладу: " + Nsum);
```

За допомогою **функції *prompt()*** в браузері виводиться діалогове вікно з пропозицією ввести деяке значення. Другий аргумент цієї функції – значення, яке буде використовуватися за замовчуванням. Однак функція *prompt()* повертає рядок, тому цей рядок потрібно перетворити в число, щоб виконувати з ним операції.

Після відкриття веб-сторінки з підключеним js-кодом, наведеним вище, в браузері можна побачити запрошення до вводу суми вкладу. Далі відобразиться аналогічне вікно для вводу процентної ставки, після чого програма отримує дані, перетворює їх в числа і виконує підрахунки.

5.1.6. Масиви

Для роботи з наборами даних призначені масиви. Для **створення масиву** застосовується вираз *new Array()*:

var myArray = new Array();

Також існує більш короткий спосіб оголошення масиву:

var myArray = [];

У цьому випадку створюється порожній масив.

Внесемо дані в масив:

var people = ["George", "Yuzik", "Alex"];

Для **звернення до окремих елементів масиву** використовуються індекси. Відлік починається з нуля, тобто перший елемент має індекс 0, а останній – 2:

```
var people = ["George", "Yuzik", "Alex"];
console.log(people[0]); // George
var p = people[2];
console.log(p); // Alex
```

Якщо звернутися до елемента за індексом, який більший, ніж розмір масиву, то отримаємо значення *undefined*:

```
var people = ["George", "Yuzik", "Alex"];
console.log(people[7]); // undefined
```

За індексом здійснюється й задання значень для елементів масиву:

```
var people = ["George", "Yuzik", "Alex"];
console.log(people [0]); // George
people[0] = "Ivo";
console.log(people[0]); // Ivo
```

Причому, на відміну від інших мов, напр. C# або Java, **можна ініціалізувати елемент, який попередньо не був визначений:**

```
var people = ["George", "Yuzik", "Alex"];
console.log(people[7]);
  // undefined – в масиві тільки три елементи
people[7] = "Ivo";
console.log(people[7]); // Ivo
```

На відміну від інших мов програмування, в JavaScript **масиви не строго типізовані**, тобто один масив може зберігати дані різних типів:

var objs = ["Alex", 44, 10.79, true];

console.log(objs);

Масиви можуть бути одновимірними та багатовимірними. Кожен елемент в багатовимірному масиві може являти собою окремий масив. Досі ми розглядали тільки одновимірні масиви.

Створимо **багатовимірний масив**:

var arr = [[5, 45, 2], [67, 4, 45]]; // двовимірний масив

Оскільки масив *arr* двовимірний, він являє собою просту таблицю. Кожен його елемент може представляти окремий масив. Розглянемо ще один приклад двовимірного масиву:

```
var people = [
    ["Alex", 33, false],
    ["George", 44, true],
    ["Yuzik", 55, false]
];
console.log(people[0]); // ["Alex", 33, false]
console.log(people[1]); // ["George", 44, true],
```

Щоб звернутися до окремого елемента масиву, також використовується індекс:

```
var AlexInfo = people[0];
console.log(AlexInfo[0]); // Alex
console.log(AlexInfo[1]); // 33
```

Але тепер змінна *AlexInfo* являє собою масив.

Щоб отримати елемент всередині вкладеного масиву, потрібно використовувати його другу розмірність:

```
console.log(people[0][0]); // Alex
console.log(people[0][1]); // 33
```

Тобто, якщо візуально двовимірний масив можна зобразити у вигляді таблиці, то елемент *people[0][1]* – комірка таблиці, яка знаходиться на перетині 0-вого рядка та 1-ого стовпця (перша розмірність – 0-й рядок, друга розмірність – 1-й стовпець).

Також можна виконати присвоєння:

```
var people = [
    ["Alex", 33, false],
    ["George", 44, true],
    ["Yuzik", 55, false]
];
```

people[0][1] = 66; // присвоюємо значення
console.log(people[0][1]); // 66
people[1] = ["Ivo", 22, false]; // присвоюємо масив
console.log (people[1][0]); // Ivo

При створенні багатовимірних масивів не має обмеження тільки двовимірними, також можна створювати **масиви довільних розмірностей**:

var arr = [];
arr[0] = []; // тепер arr – двовимірний масив
arr[0][0] = []; // тепер arr – тривимірний масив
arr[0][0][0] = 66; // 1-ий елемент 3-вимірного масиву = 66
console.log (arr[0][0][0]); // 66

Щоб взнати **довжину масиву**, використовують властивість *length*. Фактично, довжиною масиву є індекс останнього елемента з додаванням одиниці. Наприклад:

var people = new Array(); // в масиві 0 елементів
people[0] = "George";
people[1] = "Yuzik";
people[4] = "Alex";
for (var i = 0; i < people.length; i++)
  console.log (people[i]);

Результат виводу в консолі браузера:

George
Yuzik
undefined
undefined
Alex

Копіювання масиву може бути **поверхневим**, його ще називають **неглибоким** (*shallow copy*) і **глибоким** (deep copy). Неглибоке копіювання визначається присвоєнням змінній значення іншої змінної, яка зберігає масив:

*var **users** = ["George", "Yuzik", " Alex"];*
console.log(users); // ["George", "Yuzik", " Alex"]
*var **people = users**; // неглибоке копіювання*
***people[1] =** "Ivo"; // змінюємо другий елемент*
*console.log(**users**); // ["George", "Ivo", " Alex"]*
*console.log(**people**); // ["George", "Ivo", " Alex"]*

У даному випадку, **змінна *people* після копіювання** буде вказувати на той же масив, що й змінна *users*. Тому при зміні елементів в *people* зміняться елементи і в *users*, оскільки це фактично, один і той же масив. Така ситуація не завжди є бажаною, наприклад, якщо потрібно, щоб після копіювання змінні вказували на окремі масиви. У цьому випадку можна використовувати **глибоке копіювання** за допомогою методу *slice()*:

```
var users = ["George", "Yuzik", " Alex"];
console.log(users); // ["George", "Yuzik", " Alex"]
var people = users.slice(); // глибоке копіювання
people[1] = "Ivo"; // змінюємо другий елемент
console.log(users); // ["George", "Yuzik", " Alex"]
console.log(people); // ["George", "Ivo", " Alex"]
```

У цьому випадку, після копіювання змінні будуть вказувати на різні масиви, і можна їх змінювати незалежно один від іншого.

Методи для роботи з масивами:

- *push()* додає елемент в кінець масиву;
- *pop()* видаляє останній елемент масиву;
- *shift()* повертає і видаляє перший елемент масиву;
- *concat()* повертає новий масив, утворений об'єднанням довільної кількості масивів та окремих значень, які перераховані в методі як аргументи;
- *join()* об'єднує всі елементи масиву в один рядок;
- *unshift()* додає новий елемент в початок масиву;
- *splice()* видаляє всі елементи масиву, починаючи із заданого індексу;
- *sort()* сортує масив;
- *every()* перевіряє, чи всі елементи відповідають заданій умові;
- *reverse()* змінює порядок розташування елементів масиву на обернений;
- *indexOf()* і *lastIndexOf()* повертають індекс першого та, відповідно, останнього входження шуканого елемента в масиві; в протилежному випадку повертають -1;

• *filter()* приймає функцію умови й повертає масив тих елементів, які відповідають вказаній умові;

• *some()* перевіряє, чи відповідає хоча б один елемент умові, і в цьому випадку, повертає *true*; якщо елементів, що відповідають умові, в масиві немає, то метод повертає *false*;

• *forEach()* і *map()* здійснюють перебір елементів і виконують з ними певні операції.

5.1.7. Умовні конструкції

Умовні конструкції дозволяють виконати ті чи інші дії в залежності від певних умов.

Конструкція *if* перевіряє задану умову, якщо ця умова істинна, то виконує зазначені дії.

Загальна форма конструкції *if*: ***if (умова) дії***. Наприклад:

```
var x = 100;
if (x > 50) alert ("значення більше 50");
```

У цьому випадку в конструкції *if* використовується умова: *x>50*. Якщо ця умова повертає *true*, тобто змінна *x* має значення більше 50, то браузер відображає повідомлення. Якщо ж значення *x* менше дорівнює 50, то жодних дій виконуватися не буде. Якщо у випадку правильності умови необхідно виконати набір операторів, то їх потрібно розмістити у фігурних дужках:

```
var x = 100;
if (x > 50) {
   var mes = "значення більше 50";
   alert(mes);
}
```

Умови можуть бути складеними:

```
if (x < 150 && y > 180)
```

Конструкція *if* дозволяє **перевірити наявність додатного або від'ємного значення**. Наприклад:

```
var x = 89;
if (x) {
   // дії
}
```

Якщо змінна *x* має значення, додатне або від'ємне, не 0, то в умовній конструкції вираз *(x)* поверне значення *true*.

Для перевірки значення змінної використовують альтернативний варіант – **перевіряють на значення *undefined***:

```
if (typeof x != "undefined") {
   // дії
}
```

У конструкції *if* також можна використовувати **блок *else***, який містить оператори, які виконуються, якщо умова в блоці *if* хибна, тобто дорівнює *false*:

```
var a = 30;
if (a >= 40) {  alert("умова істинна");
} else { alert("умова хибна");
}
```

За допомогою **конструкції *else if*** можна додати альтернативну умову до блоку *if*:

```
var x = 3000;
if (x <2000) { alert("x < 2000"); }
else if (x >= 2000 && x <= 4000) {
        alert("2000 <= x <= 4000"); }
else { alert("x > 4000"); }
```

У цьому випадку виконається блок *else if*. При необхідності можна використовувати кілька блоків *else if* з різними умовами:

```
var x = 3000;
if (x <2000) { alert("x < 2000"); }
else if (x >= 2000 && x < 3000) { alert("2000 <= x < 3000"); }
else if (x >= 3000 && x < 4000) { alert("3000 <= x < 4000"); }
else { alert("x >= 4000"); }
```

У JavaScript будь-яка змінна може використовуватися в умовних виразах, але не будь-яка змінна типу *boolean*. У зв'язку із цим виникає **питання, що поверне та чи інша змінна – значення *true* або *false*?** Це залежить від типу даних змінної:

- ***undefined*** повертає *false*;
- ***null*** повертає *false*;
- ***boolean*** повертає *false*, якщо змінна дорівнює *false*, і відповідно, якщо змінна дорівнює *true*, то повертає *true*;
- ***number*** повертає *false*, якщо число дорівнює *0* або *NaN* (*Not a Number*), в інших випадках – *true*; наприклад:

```
let x = NaN;
```

```
if (x)  alert("true");
else alert("false");
  // буде виведено вікно з повідомленням false
```

• ***string*** повертає *false*, якщо змінна – порожній рядок, тобто довжина рядка *0*, в інших випадках – *true*:

```
var y = ""; // false –  тому що змінна y – порожній рядок
var z = "JavaScript"; // true – рядок не порожній
```

• ***object*** – завжди повертає *true*:

```
var person = {name: "Alex"}; // true
var animal = {}; // true
```

Конструкція *switch..case* є альтернативою використання конструкції *if…else if…else …*, також дозволяє опрацювати одночасно кілька умов:

```
var x = 300;
switch(x){
   case 100:
      console.log("x = 100"); break;
   case 200:
      console.log("x = 200"); break;
   case 300:
      console.log("x = 300"); break;
   default:
      console.log("Невідоме значення"); break;
}
```

Після ключового слова *switch* у дужках вказується вираз. Значення цього виразу послідовно порівнюється зі значеннями, розміщеними після операторів *case*. Якщо відбудеться збіг, то будуть виконуватися оператори того блоку *case*, зі значенням якого наявний збіг. У кінці кожного блоку *case* розташовується **оператор *break***, щоб уникнути виконання інших блоків. Щоб обробити ситуацію, коли збіг не відбудеться, конструкцію доповнюють **блоком *default***.

Тернарна операція приймає три операнди і має такий синтаксис:

[перший операнд – умова] ? [другий операнд]: [третій операнд];

Залежно від умови, тернарна операція повертає другий або третій операнд: якщо умова дорівнює *true*, то повертає другий операнд; якщо – *false*, то третій. Наприклад:

```
var n = 1;
var m = 2;
var result = n < m ? n + m : n - m;
console.log(result); // 3
```

Якщо значення змінної *n* строго менше значення змінної *m*, то змінна *result* буде дорівнювати *n+m*, інакше *n-m*.

5.1.8. Цикли

Цикли дозволяють, залежно від певних умов, виконувати деяку дію багато разів. У JavaScript є такі види циклів:

- *for*;
- *for..in*;
- *for..of*;
- *while*;
- *do..while*.

Розглянемо детально кожний вид циклів:

1. Цикл *for* має таке формальне визначення:

```
for([ініціалізація лічильника]; [умова]; [зміна лічильника]) {
   // дії
}
```

Наприклад, можна використати цикл *for* для перебору елементів масиву:

```
var users = ["George", "Alex", "Yuzik", "Ivo"];
for (var i = 0; i < users.length; i++) {
   console.log(users[i]);
}
```

Перша частина оголошення циклу: ***var i = 0*** створює і ініціалізує лічильник циклу – змінну *i*. Перед виконанням циклу її значення буде дорівнювати 0.

Друга частина – умова, при якій буде виконуватися цикл: ***i < users.length***. У цьому випадку цикл буде виконуватися, поки значення змінної *i* не дорівнює довжині масиву *people*. Отримати довжину масиву можна за допомогою властивості *length*: *users.length*.

Третя частина – збільшення лічильника на одиницю.

Оскільки в масиві 4 елементи, то блок циклу спрацює 4 рази, поки значення *i* не стане рівним *users.length* (тобто 4). Кожного разу змінна *i* буде збільшуватися на 1. Кожний повтор циклу називається **ітерацією**. Таким чином, у цьому випадку, виконається 4 ітерації.

За допомогою виразу *users[i]* можна звернутися до елемента масиву, в даному випадку – для його подальшого виведення в браузері. Необов'язково збільшувати лічильник циклу на одиницю, можна виконувати з ним інші дії, наприклад, зменшувати на одиницю тощо.

2. Цикл *for..in* призначений для перебору масивів і об'єктів. Його формальне визначення:

```
for (індекс in масив) {
   // дії
}
```

Наприклад, переберемо елементи масиву:

```
var users = ["George", "Alex", "Yuzik", "Ivo"];
for (var index in users) {
     console.log(users[index]);
}
```

3. Цикл *for ... of* схожий на цикл *for ... in* і призначений для перебору колекцій, наприклад масивів:

```
var people = ["George", "Alex", "Yuzik", "Ivo"];
for (let x of people)
   console.log(x);
```

Поточний елемент перебору колекції розміщується в змінну *x*, значення якої виводиться на консоль.

4. Цикл *while* виконується поки задана умова істинна. Його формальне визначення:

```
while (умова) {
   // дії
}
```

Виведемо на консоль елементи масиву в циклі *while*:

```
var users = ["George", "Alex", "Yuzik", "Ivo"];
var i = 0;
while (i < users.length) {
   console.log(users[i]);
```

```
    i++;
}
```

Цикл *while* буде виконуватися, поки значення змінної *i* не дорівнює довжині масиву.

5. У циклі *do…while* спочатку виконується тіло циклу – набір операторів в блоці *do*, і після – перевірка умови в блоці *while*. Поки ця умова істинна, цикл буде повторюватися. Наприклад:

```
var j = 1;
do { console.log(j * j); j++;
} while (j < 10)
```

У цьому випадку, тіло циклу виконується 9 разів, поки змінна *j* не дорівнює 10.

Цикл *do…while* гарантує хоча б одноразове виконання дій, навіть якщо умова циклу хибна.

Іноді буває необхідно **вийти із циклу до його завершення**. У цьому випадку можна скористатися **оператором *break***:

```
var numbers = [1, 2, 3, 4, 5, 12, 17, 6, 7];
for (var i = 0; i < numbers.length; i++) {
    if (numbers[i] > 10) break;
    document.write(numbers[i] + "</br>");
}
```

Цей цикл перебирає всі елементи масиву, проте останні чотири елементи не будуть виведені на сторінці, оскільки перевірка *if (numbers[i]>10)* перерве виконання циклу за допомогою оператора *break*, коли перебір масиву дійде до елемента 12.

Якщо потрібно **пропустити ітерацію циклу, але не виходити з нього**, то можна застосовувати **оператор *continue***:

```
var numbers = [1, 2, 3, 4, 5, 12, 17, 6, 7];
for (var i = 0; i < numbers.length; i++){
    if (numbers[i] > 10) continue;
    document.write(numbers[i] + "</br>");
}
```

У цьому випадку, числа, більші за 10, не будуть виведені на сторінці.

5.2. Функціональне програмування

5.2.1. Функції

Функція – це набір операторів, що виконують певні дії або обчислюють певне значення. Синтаксис визначення функції:

```
function імя_функції([параметр [...]]) {
    // Оператори
}
```

Визначення функції починається з ключового слова *function*, після якого вказують ім'я функції. **Ім'я функції** підпорядковується тим же правилам, що й ім'я змінної, тобто може містити тільки букви, цифри, символи підкреслення та долара ($), і має починатися з букви, символу підкреслення або долара.

Після імені функції в дужках перераховуються **параметри**. Якщо функція не має параметрів, то дужки залишаються порожніми. Далі в фігурних дужках – **тіло функції**, що являє собою набір операторів. Визначимо найпростішу функцію:

```
function display() {
    document.write("функція в JavaScript");
}
```

Ця функція називається *display()*. Вона не має жодних параметрів і все, що вона робить – це виводить на веб-сторінці рядок, який розташовується в лапках. Однак визначення функції не означає її **запуск та виконання**. Для цього функцію потрібно викликати:

```
function display() {
    document.write("функція в JavaScript");
}
display(); //виклик функції
```

Необов'язково давати функцій ім'я. Можна використовувати **анонімні функції**:

```
var display = function() {// визначення функції
    document.write("функція в JavaScript");
}
display();
```

У коді вище визначено змінну *display* і їй присвоєно посилання на функцію. Далі функція викликається по імені

змінної.

Також можна **динамічно присвоювати функції змінним**:

```
function Morning() { document.write("Добрий ранок!"); }
function Evening() { document.write("Добрий вечір!"); }
var message = Morning;
message(); // Добрий ранок!
message = Evening;
message(); // Добрий вечір!
```

Розглянемо **передачу параметрів у функцію**:

```
function display(y) {// визначення функції
   var x = y * y;
   document.write (y + " в квадраті = " + x);
}
display(55); // виклик функції
```

Функція *display* має один параметр *y*, тому при виклику функції потрібно передати для нього значення, наприклад число *55*.

Якщо функція має кілька параметрів, то за допомогою **spread-оператора** (який має вигляд трьох крапок: **...**) можна передати набір значень для цих параметрів з масиву:

```
function sum(x, y, z) {
   let w = x + y + z; console.log(w);
}
sum(11, 9, 5);  //25
let numbers = [3, 1, 60];
sum(... numbers);  //64
```

У другому випадку, у функцію передаються числа з масиву *numbers*. Але щоб передати масив не як одне значення, а саме числа з цього масиву, використовується *spread-оператор (три крапки ...)*.

Функція може приймати **багато параметрів, при цьому частина з них або всі можуть бути необов'язковими**. Якщо для параметрів не передаються значення, то, за замовчуванням, їх значення «*undefined*». Наприклад:

```
function display(a, b) {
   if (a === undefined) a = 2;
   if (b === undefined) b = 3;
   let c = a * b;
```

```
    console.log(c);
}
display(); // 6
display(1); // a=1, b= undefined, результат 3
display(1, 4) // 4
```

Функція *display* має два параметри. При виконанні функції перевіряється наявність їх значення. При цьому, викликаючи функцію, необов'язково передавати для цих параметрів значення. Для **перевірки наявності значення параметрів** використовується порівняння зі значенням *undefined*.

Є **інший спосіб задання значення параметрів за замовчуванням**:

```
function display(a = 2, b = 3) {
    let c = a * b;  console.log(c);
}
```

Якщо параметрам *a* і *b* не передаються значення, то вони будуть мати значення 2 і 3 відповідно. Такий спосіб більш лаконічний і інтуїтивний, ніж порівняння з *undefined*. При цьому значення параметра, за замовчуванням, може бути похідним, тобто являти собою вираз:

```
function display(a = 2, b = 3 + a) { … }
```

В даному випадку, значення параметра *b* залежить від значення параметра *a*.

При необхідності можна отримати всі передані параметри через глобально доступний **масив *arguments***:

```
function display() {
   var product = 1;
   for (var i = 0; i < arguments.length; i++)
      product *= arguments[i];
   console.log(product);
}
display(3); // 3
display(3, 5, 3) // 45
display(2, 2, 3) // 12
```

При цьому не важливо, що при визначенні функції не вказані жодні параметри, все одно їх можна передати та отримати їх значення через масив *arguments*.

5.2.2. Область видимості змінних

Усі змінні в JavaScript мають свою **область видимості**, в межах якої вони діють. **Всі змінні, оголошені поза функціями, є глобальними**:

```
<script>
  var x = 5;
  let b = 8;
  function printSquare() { var y = x*x; console.log(y); }
  printSquare(); //25
</script>
```

Тут змінні *x* і *b* глобальні. Вони доступні з будь-якого місця програми. А змінна *y* не є глобальною, оскільки вона визначена всередині функції.

Змінна, визначена всередині функції, називається локальною. Таким чином, змінна *y*, визначена в попередньому прикладі, локальна. Локальні змінні існують тільки в межах функції. За межами функції їх неможливо використовувати, оскільки їх там не існує. Коли функція завершує свою роботу, всі змінні, визначені в функції, знищуються, тобто перестають існувати.

Якщо є дві змінні – одна глобальна, а інша локальна, які мають однакове ім'я, то при використанні цього імені всередині функції використовуватиметься та змінна, яка визначена безпосередньо в функції. Тобто локальна змінна "приховає" глобальну.

При використанні оператора *let* кожен блок коду визначає нову область видимості, в якій існує змінна. Наприклад, можна одночасно визначити змінну на рівні блоку і на рівні функції:

```
let z = 10;
function printZ() {
   let z = 20;
   { let z = 30; console.log("Блок:", z); }
   console.log("Функція:", z);
}
printZ();
console.log("Глобально:", z);
```

Результуючий вивід:

```
Блок: 30
```

Функція: 20
Глобально: 10

Тут всередині функції *printZ* визначено блок коду, в якому визначена змінна z: *let z = 30* приховує глобальну змінну *let z = 10* і змінну z, визначену на рівні функції: *let z = 20*. У реальній програмі блок може бути представлений вкладеною функцією, блоком циклу *for* або конструкцією *if*. Але в будь-якому випадку такий блок визначає нову область видимості, поза якою змінна, яка в ньому визначена, не існує.

Якщо замість оператора *let*, використати **оператор *var*** у всіх трьох випадках, то результат буде наступним:

Блок: 30
Функція: 30
Глобально: 30

А у такому випадку:

```
function printZ() {
   let z = 20;
   { var z = 30; console.log("Блок:", z); }
   console.log("Функція:", z);
}
displayZ();
```

буде помилка: «*Uncaught SyntaxError: Identifier 'z' has already been declared*».

Все, що відноситься до оператора *let*, відноситься і до **оператора *const***, який дозволяє визначити константи. Блоки коду задають область видимості констант, а константи, визначені у вкладених блоках коду, приховують зовнішні константи з тим самим ім'ям.

Якщо **не використовуються ключові слова** (*let*, *var*, *const*) при визначенні змінної у функції, то така змінна буде глобальною. Наприклад:

```
function bar() { foo = "25"; }
bar();
console.log(foo); // 25
```

Визначення глобальних змінних у функціях може призводити до потенційних помилок. Щоб їх уникнути використовується **строгий режим** (strict mode). Встановити такий режим можна двома способами:

• додати вираз "*use strict*" в початок коду JavaScript, тоді **strict mode** буде застосовуватися для всього коду;

• додати вираз "*use strict*" в початок тіла функції, тоді **strict mode** буде застосовуватися тільки для цієї функції.

Наприклад:

```
"use strict";
function bar(){
   foo = "25";   // Uncaught ReferenceError: foo is not defined
}
bar();
console.log(foo);
```

5.2.3. Замикання

Замикання являє собою конструкцію, коли функція, створена в одній області видимості, запам'ятовує своє **лексичне середовище** (оточення). Замикання технічно включає **три компоненти**:

• зовнішня функція, яка визначає деяку область видимості і в якій визначені деякі змінні – лексичне оточення;

• змінні (лексичне оточення), які визначені у зовнішній функції;

• вкладена функція, яка використовує ці змінні.

Розглянемо замикання на простому прикладі:

```
function outer() {
   let a = 10;
      //змінна a – лексичне оточення для функції inner
   function inner() {a++; console.log (a); };
   return inner;
}
let fn = outer();
      // fn = inner, оскільки outer повертає функцію inner
// викликаємо внутрішню функцію inner
fn(); // 11
fn(); // 12
fn(); // 13
```

Отже, функція *outer* задає область видимості, в якій визначені внутрішня функція *inner* і змінна *a*. Змінна *a* являє

собою **лексичне оточення для функції *inner***. У функції *inner* збільшуємо змінну *a* і виводимо її значення на консоль. Функція *outer* повертає функцію *inner*. Далі викликаємо функцію *outer*:

```
let fn = outer();
```

Оскільки функція *outer* повертає функцію *inner*, то змінна *fn* буде зберігати посилання на функцію *inner*. При цьому **функція *inner* запам'ятала своє оточення,** тобто зовнішню змінну *a*. Далі фактично три рази викликається функція *inner()*, і можна побачити, що змінна *a*, яка визначена поза функцією *inner*, збільшується.

Таким чином, незважаючи на те, що **змінна *a* визначена поза функцією *inner***, ця функція запам'ятала своє оточення (змінну *a*) і може його використовувати, недивлячись на те, що вона викликається поза функцією *outer*, в якій була визначена. В цьому суть замикань.

5.2.4. Самовикликаючі, рекурсивні та стрілочні функції

Зазвичай визначення функції відділяється від її виклику: спочатку визначається функція, а потім її виклик. Але в JavaScript це необов'язково. Можна створити такі функції, які будуть викликатися при їх визначенні. Такі функції ще називають **Immediately Invoked Function Expression** (IIFE). Приклади:

```
(function() { console.log("Привіт світ"); }());
(function(n) {
   var result = 1;
   for (var i = 1; i <= n; i++)
      result *= i;
   console.log("Факторіал числа "+n+" дорівнює " + result);
}(4));// (4) – це виклик функції
```

Серед функцій також можна виділити **рекурсивні функції**. Їх суть полягає в тому, що функція викликає саму себе.

Наприклад, розглянемо функцію, яка знаходить факторіал числа:

```
function Factorial(n) {
   if (n === 1) {
      return 1;
   } else {
```

```
        return n * Factorial(n - 1);
    }
}
var result = Factorial(4);
console.log(result); // 24
```

Функція *Factorial()* повертає значення 1, якщо параметр *n* дорівнює 1, або повертає як результат виклик цієї ж функції *Factorial* зі значенням параметра *n-1*. Наприклад, при передачі числа 4, утворюється такий ланцюжок викликів:

```
var result = 4 * Factorial (3);
var result = 4 * 3 * Factorial (2);
var result = 4 * 3 * 2 * Factorial (1);
var result = 4 * 3 * 2 * 1; // 24
```

Варіантом подання функцій є **стрілочні функції (arrow functions)**, які являють собою скорочений запис звичайних функцій. Стрілочні функції записуються за допомогою знака стрілки (=>), перед яким у круглих дужках розміщуються параметри функції, а після у фігурних дужках – тіло функції. Наприклад:

```
let sum = (x, y) => { return x + y;}
let a=sum(4,5); // 9
let b = sum(10, 5); // 15
```

У цьому випадку, функція *(x, y) => { return x + y;}* виконує додавання двох чисел та присвоєння результату змінній *sum*. Функція має два параметри – *x* і *y*. Її тіло – операція додавання значень цих параметрів і, відповідно, функція повертає це значення.

Якщо в тілі функції єдиним оператором є *return вираз*; то допускається опускати фігурні дужки і ключове слово *return*. Тому вищенаведену функцію обчислення суми двох чисел можна рівносильно записати так:

```
let sum = (x, y) => x + y;
let z=sum(5,6); // 11
```

Оскільки після стрілки слідує вираз, що являє собою числове значення суми чисел, то функція повертає це значення. Можна через змінну *sum* викликати цю функцію і отримати її результат в змінну *z*.

Якщо тілом функції є один єдиний вираз, який нічого не повертає і лише виконує певну дію, то фігурні дужки, що визначають тіло функції, також можна опускати:

```
let sum = (x, y) => console.log (x + y);
sum(4, 5); // 9
sum(10, 5); // 15
```

У цьому випадку, функція нічого не повертає і, відповідно, функція *sum* також не повертає результату.

Якщо функція має один параметр, то круглі дужки, в яких мав би розміщуватися цей параметр, можна опускати, тобто запис

```
var square = (n) => { let result = n * n; return result; }
console.log(square(5)); // 25
```

рівносильний наступному:

```
var square = n => n * n;
console.log(square(5)); // 25
```

Якщо стрілочна функція не має жодних параметрів, то ставляться порожні дужки:

```
var hello = () => console.log("Hello World");
hello(); // Hello World
```

При цьому варто пам’ятати, що якщо тіло функції містить набір операторів, то вони розміщаються у фігурних дужках:

```
var square = n => { let result = n*n; return result; }
console.log(square(5)); // 25
```

Розглянемо окремо випадок, коли **стрілочна функція повертає об’єкт:**

```
let person = (personName, personAge) =>
 ({name: personName, age: personAge});
let george = person("George", 34);
let yuzik = person("Yuzik", 25);
console.log(george.name, george.age); // George 34
console.log(yuzik.name, yuzik.age); // Yuzik 25
```

Об’єкт також визначається за допомогою фігурних дужок, але при цьому він береться в круглі дужки.

5.2.5. Перевизначення функцій

Функції в JavaScript можна перевизначати. Перевизначення відбувається за допомогою присвоєння анонімної функції змінній, яка називається так само, як і функція, яка перевизначається:

```
function display() {
   console.log("Доброго ранку");
   display = function() { console.log("Добрий день"); }
}
display(); // Доброго ранку
display(); // Добрий день
```

При першому виклику функції спрацьовує основний блок операторів функції: виводиться повідомлення «*Доброго ранку*» та відбувається перевизначення функції. Тому при всіх наступних викликах функції спрацьовує її перевизначена версія, і консоль виведе повідомлення «*Добрий день*».

При перевизначенні функції потрібно враховувати деякі нюанси. А саме, спробуємо **присвоїти посилання на функцію змінній і через цю змінну викликати функцію**:

```
function display() {
   console.log("Доброго ранку");
   display = function () { console.log("Добрий день"); }
}
// присвоєння посилання на функцію до перевизначення
var displayMessage = display;
display(); // Доброго ранку
display(); // Добрий день
displayMessage(); // Доброго ранку
displayMessage(); // Доброго ранку
```

Отже, змінна *displayMessage* отримує посилання на функцію *display* до її перевизначення. Тому при виклику *displayMessage()* буде викликатися неперевизначена версія функції *display*.

Але у випадку, якщо змінна *displayMessage* визначена вже після виклику функції *display*, то змінна *displayMessage* буде вказувати на перевизначену версію функції *display*:

```
display(); // Доброго ранку
display(); // Добрий день
var displayMessage = display;
```

displayMessage(); // Добрий день
displayMessage(); // Добрий день

Таким чином, перевизначення функції *display* відбувається тільки шляхом виклику самої функції *display*.

5.2.6. Процес Hoisting

Hoisting – це процес доступу до змінних до їх визначення. Можливо, ця концепція виглядає дещо дивно, але вона пов'язана з роботою компілятора JavaScript. Компіляція коду відбувається в два проходи. **При першому проході** компілятор отримує всі оголошення змінних, всі ідентифікатори. При цьому жодний код не виконується, методи не викликаються. **При другому проході** відбувається виконання коду. Якщо змінна визначена після безпосереднього використання, помилки не виникне, тому що при першому проході компілятору вже відомі всі змінні. Тобто відбувається ніби "підняття" коду з визначенням змінних і функцій вгору до їх безпосереднього використання. "Підняття" перекладається англійською як **hoisting**, тому цей процес так і називається. **Змінні, які потрапляють під hoisting**, отримують значення *undefined*.

Наприклад, розглянемо найпростіший код:

console.log(foo);

Його виконання викличе помилку *ReferenceError: foo is not defined*. Додамо визначення змінної:

console.log(foo); // undefined
var foo = "George";

У цьому випадку, консоль виведе значення *undefined*. При першому проході компілятор дізнається про існування змінної *foo*. Вона отримує значення *undefined*. При другому проході викликається метод *console.log(foo)*. Розглянемо інший приклад:

*var z = x*y;*
var x = 5;
var y = 2;
console.log(z); // NaN

Тут виникає така ж ситуація. Змінні *x* і *y* використовуються до визначення. За замовчуванням, їм присвоюється значення *undefined*. Якщо помножити *undefined* на *undefined*, то отримаємо *Not a Number (NaN)*. Це відноситься і до

використання функцій. Можна спочатку викликати функцію, а потім вже її визначити:

```
display();
function display() { console.log( "Hello Hoisting"); }
```

Тут функція *display* вдало спрацює, незважаючи на те, що вона визначена після виклику. Але від цієї ситуації потрібно відрізняти той **випадок, коли функція визначається у вигляді змінної**:

```
display();
var display = function() { console.log("Hello Hoisting"); }
```

В останньому випадку, буде помилка *TypeError: display is not a function*. При першому проході компілятор також отримає змінну *display* і присвоїть їй значення *undefined*. При другому проході, коли треба буде викликати функцію, на яку буде посилатися ця змінна, компілятор побачить, що викликати нічого: змінна *display* поки ще дорівнює *undefined*.

5.2.7. Передача параметрів за значенням і за посиланням

Рядки, числа, логічні значення передаються в функцію за значенням. Іншими словами, при передачі значення в функцію, функція отримує копію цього значення. Розглянемо, що це означає в практичному плані:

```
function change(x) {
   x = 2*x;
   console.log("x змінено: ", x);
}
var n = 10;
console.log("n до зміни: ", n); // n до зміни: 10
change(n); // x змінено: 20
console.log("n після зміни: ", n); // n після зміни: 10
```

Функція *change* отримує як параметр певне число та збільшує його в два рази. При виклику функції *change* їй передається значення числа *n*. Однак після виклику функції можна побачити, що число *n* не змінилося, хоча у самій функції задано збільшення значення параметра. Тому що при виклику функція *change* отримує **копію значення змінної *n***. І будь-які зміни з цією копією жодним чином не зачіпають саму змінну *n*.

Об'єкти і масиви передаються за посиланням, тобто функція отримує сам об'єкт або масив, а не їх копію.

```
function change(person) {
   person.name = "Yuzik";
}
var george = { name: "George" };
console.log("до зміни: ", george.name); //до зміни: George
change(george);
console.log("після зміни: ", george.name); //після зміни: Yuzik
```

У цьому випадку, функція *change* отримує об'єкт і змінює його властивість *name*. У підсумку, після виклику функції змінився оригінальний об'єкт *george*, який передавався в функцію. **Те ж саме стосується масивів**:

```
function change(array) { array[0] = 8; }
function completeChange(array) { array = [9, 8, 7]; }
var nums = [1, 2, 3];
console.log("до зміни: ", nums); // до зміни: [1, 2, 3]
change(nums);
console.log("після зміни: ", nums); // після зміни: [8, 2, 3]
completeChange(nums);
console.log("після повної зміни: ", nums);
  //після повної зміни: [8, 2, 3]
```

Однак при спробі перевизначити об'єкт або масив повністю, оригінальні значення не зміняться.

5.3. Об'єктно-орієнтоване програмування в JavaScript

5.3.1. Загальні відомості

Об'єктно-орієнтоване програмування на сьогоднішній день є однією з основних парадигм в розробці додатків, і JavaScript надає можливість використовувати всі переваги ООП. У той же час об'єктно-орієнтоване програмування в JavaScript має свої особливості.

Об'єктно-орієнтований js-код може бути використаний, наприклад, якщо в програмі потрібно описати сутність "людина", яка має свої характеристики: ім'я, вік, стать і так далі, і природно, що не можна уявити сутність "людина" у вигляді числа або рядка. Потрібно набір рядків і/або чисел, щоб

належним чином описати “людину”. В цьому плані людина буде виступати як складна комплексна структура, що складається з окремих властивостей: вік, зріст, ім’я, прізвище і т.д.

Для роботи з подібними структурами в JavaScript використовуються **об’єкти**. Кожен об’єкт може зберігати **властивості,** які описують його стан, і **методи**, які описують його поведінку.

Створення нового об’єкта:

var person = new Object();

var person = {};

Після створення об’єкта можна **визначити в ньому властивості**. Щоб визначити властивість, потрібно після імені об’єкта через крапку вказати ім’я властивості і присвоїти їй значення:

```
var person = {};
person.name = "George";
person.age = 26;
```

У даному випадку, оголошуються дві властивості *name* і *age*, яким присвоюються значення. Після цього можна використовувати ці властивості, наприклад, вивести в консолі їх значення:

```
console.log(person.name); // George
console.log(person.age); // 26
```

Також можна визначити властивості при створенні об’єкта:

```
var person = {
   name: "George",
   age: 26
};
```

У цьому випадку, для присвоєння значення властивості використовується символ двокрапки, а після визначення значення ставиться кома (а не крапка з комою).

Крім того, доступний скорочений спосіб визначення властивостей:

```
var name = "George";
var age = 34;
var person = {name, age};
```

Методи об'єкта визначають його поведінку, тобто дії, які він виконує або які виконується з ним. Фактично методи являють собою функції. Наприклад, визначимо метод, який виводить ім'я і вік людини:

```
var person = {};
person.name = "George";
person.age = 26;
person.display = function() {
   console.log(person.name);
   console.log(person.age);
};
person.display();// виклик методу
```

Як і у випадку з функціями, методи спочатку визначаються, а потім вже викликаються. Також методи можуть визначатися безпосередньо при визначенні об'єкта.

Щоб звернутися до властивостей або методів об'єкта всередині цього об'єкта, використовується **ключове слово *this***, яке означає посилання на поточний об'єкт.

Можна використовувати скорочений спосіб визначення методів, коли слово *function* опускається:

```
var person = {
   name: "George",
   age: 26,
   display() { console.log(this.name, this.age); },
};
person.display();// виклик методу
```

Існує також **альтернативний спосіб визначення властивостей і методів** за допомогою синтаксису масивів:

```
var person = {};
person ["name"] = "George";
person ["age"] = 26;
person ["display"] = function() {
   console.log(person.name);
   console.log(person.age);
};
person ["display"](); //виклик методу
person.display(); //виклик методу спрацює так само
```

При **звертанні до таких властивостей і методів** можна використовувати або нотацію крапки (*person.name*), або звернутися так: *person["name"]*. Слід зазначити, що **назви властивостей і методів** об'єкта завжди являють собою рядки. Тому можна властивість *name* визначити так:

"name": "George",

З одного боку, немає жодної різниці між цими двома визначеннями. З іншого, бувають випадки, де назви у вигляді рядків можуть допомогти. Наприклад, якщо назва властивості складається з двох слів, розділених пробілом:

"Display info": function() { …};

Можна **видаляти властивості і методи** за допомогою оператора *delete*. Як і у випадку з їх визначенням, можна видаляти властивості двома способами:

- *delete об'єкт.властивість;*
- *delete об'єкт["властивість"].*

Після видалення властивість буде не визначена, тому при спробі звернення до неї, програма поверне значення *undefined*.

Одні об'єкти можуть містити як властивості інші об'єкти. Наприклад, є об'єкт "країна", для якого можна визначити ряд властивостей. Одна з цих властивостей може означати столицю. Але для столиці також можна виділити свої властивості, наприклад назву, чисельність населення, рік заснування:

```
var country = {
   name: "Німеччина",
   language: "німецька",
   capital: { name: "Берлін", population: 3375000, year: 1237 }
};
```

Для доступу до властивостей таких вкладених об'єктів можна використовувати такий синтаксис:

- *country.capital.name;*
- *country["capital"]["population"];*
- *country.capital["year"].*

Властивості також можуть бути масивами, в тому числі масивами інших об'єктів:

var country = {

```
    name: "Швейцарія",
    languages: ["німецька", "французька", "італійська"],
    capital: {
      name: "Берн",
      population: 126598
    },
    cities: [
      {Name: "Цюріх", population: 378884},
      {Name: "Женева", population: 188634},
      {Name: "Базель", population: 164937}
    ]
};
  // виведення всіх елементів з country.languages
document.write("<h3>Офіційні мови Швейцарії</h3><br/>");
for (var i = 0; i < country.languages.length; i++)
    document.write(country.languages[i] + "<br/>");
  // виведення всіх елементів з country.cities
document.write("<h3> Міста Швейцарії </h3> <br/>");
for (var i = 0; i < country.cities.length; i ++)
    document.write(country.cities[i].Name + "<br/>");
```

В об'єкті *country* є властивість *languages*, що містить масив рядків, а також властивість *cities*, що зберігає масив однотипних об'єктів. З цими масивами можна працювати аналогічно, як і з будь-якими іншими, наприклад, перебрати за допомогою циклу *for*. При переборі масиву об'єктів кожен поточний елемент являє собою окремий об'єкт, тому можна звернутися до його властивостей і методів наступним чином: *country.cities[i].name*.

При динамічному визначенні в об'єкті нових властивостей і методів перед їх використанням буває важливо перевірити, чи є вже такі методи і властивості. Для цього в JavaScript може використовуватися **оператор *in***, що має такий вигляд:

"властивість | метод" in об'єкт;

де в лапках вказується назва властивості або методу, а після оператора *in* – назва об'єкта. Якщо властивість або метод з таким ім'ям є, то оператор повертає *true*, якщо немає – *false*. Наприклад:

var hasNameProp = "name" in person;

Оскільки **об’єкти відносяться до типу *object***, а значить, мають всі його методи і властивості, то, відповідно, також можуть використовувати метод *hasOwnProperty()*, який визначений в типі *object*:

var hasNameProp = person.hasOwnProperty('name');

Метод повертає *true* у випадку, якщо об’єкт має не успадковану властивість із зазначеним ім’ям та *false*, якщо об’єкт не має такої властивості або ця властивість успадкована від свого об’єкта прототипу.

За допомогою циклу *for..in* можна **перебрати об’єкт як звичайний масив** і отримати всі його властивості та методи:

```
for (var key in person) {
   console.log(key + ":" + person[key]);
}
```

5.3.2. Конструктори

Крім створення нових об’єктів, JavaScript надає можливість **створювати нові типи об’єктів** за допомогою конструкторів. Одним зі способів створення об’єкта є використання **конструктора типу *object***. Конструктор дозволяє визначити новий тип об’єкта. Тип являє собою абстрактний опис або шаблон об’єкта. Можна провести наступну аналогію. У нас у всіх є деяке уявлення про людину – наявність двох рук, ніг, голови, травної, нервової системи і т.д. Є деякий шаблон – цей шаблон можна назвати типом. Реально ж існуюча людина є об’єктом цього типу. Визначення типу може складатися з конструктора, методів і властивостей. Для початку визначимо конструктор:

```
function Person (n, a) {
   this.name = n;
   this.age = a;
   this.printInfo = function() {
      document.write( "Ім'я: "+this.name+ "; вік: " + this.age);
   };
}
```

Конструктор – це функція, яка призначена для визначення та задання властивостей та методів. Для звертання до властивостей і методів використовується ключове слово *this*. У

коді вище задаються дві властивості та визначається метод *printInfo*. Як правило, назви конструкторів, на відміну від назв звичайних функцій, починаються з великої літери. Після цього в програмі можна визначити об'єкт типу *Person* і використовувати його властивості та методи:

```
var george = new Person("George ", 26);
console.log(george.name); // George
george.printInfo();
```

Щоб **викликати конструктор**, тобто створити об'єкт типу *Person*, потрібно використовувати **ключове слово *new***. Аналогічно, можна визначити інші типи і використовувати їх разом.

Оператор *instanceof* дозволяє перевірити, за допомогою якого конструктора створено об'єкт. Якщо об'єкт створений за допомогою певного конструктора, то оператор повертає *true*:

```
function Person (n, a) {
   this.name = n;
   this.age = a;
   this.printInfo = function() {
      document.write( "Ім'я: " +this.name+ "; вік: " + this.age);
   };
}
function Car (title, year, price) {
   this.title = title;
   this.year = year;
   this.price = price;
}
var george = new Person("George", 26);
var isPerson = george instanceof Person;
var isCar = george instanceof Car;
console.log(isPerson); // true
console.log(isCar); // false
```

5.3.3. Прототипи

Крім безпосереднього визначення властивостей і методів, у конструкторі можна використовувати **властивість *prototype***. Кожна функція має властивість *prototype*, що визначає **прототип функції**. Тобто **властивість *Person.prototype*** задає прототип

об’єктів *Person*. І будь-які властивості і методи, які будуть визначені в *Person.prototype*, будуть загальними для всіх об’єктів *Person*.

Наприклад, після визначення об’єкта *Person* додамо до нього метод *Person.prototype.hello* і властивість *Person.prototype.maxAge*:

```
function Person(n, a) {
   this.name = n;
   this.age = a;
   this.printInfo = function() {
      document.write("Ім'я:" + this.name + "; вік:" + this.age);
   };
};
Person.prototype.hello = function () {
   document.write(this.name + " говорить: 'Привіт!' <br/>");
};
Person.prototype.maxAge = 110;
var yuzik = new Person("Юзік", 26);
yuzik.hello();
var ivo = new Person("Іво", 28);
ivo.hello();
console.log(yuzik.maxAge); // 110
console.log(ivo.maxAge); // 110
```

Отже, тут додано метод *hello* і властивість *maxAge*, і кожен об’єкт *Person* може їх використовувати. Але важливо зауважити, що значення властивості *maxAge* буде однакове для всіх об’єктів, на відміну, скажімо, від властивості *name*, яка зберігає своє значення для кожного об’єкта.

У той же час, можна визначити в об’єкті **властивість, яка буде називатися так само, як і властивість прототипу**. В цьому випадку, властивість об’єкта буде мати пріоритет над властивістю прототипу:

```
Person.prototype.maxAge = 110;
var yuzik = new Person("Юзік", 26);
var ivo = new Person("Іво", 28);
yuzik.maxAge = 99;
console.log(yuzik.maxAge); // 99
console.log(ivo.maxAge); // 110
```

При зверненні до властивості *maxAge* JavaScript спочатку шукає цю властивість серед властивостей об'єкта, і якщо її не знайдено, то звертається до властивостей прототипу. Те ж саме стосується і методів.

5.3.4. Інкапсуляція

Інкапсуляція є одним із ключових понять об'єктно-орієнтованого програмування і полягає в прихованні стану об'єкта від прямого доступу ззовні. За замовчуванням, всі властивості об'єктів публічні, тобто загальнодоступні, і до них можна звернутися з будь-якого місця програми. Але можна їх приховати від доступу ззовні, зробивши властивості локальними змінними:

```
function User(name) {
   this.name = name;
   var _age = 1; //локальна змінна
   this.displayInfo = function() {
      console.log( "Ім'я: " + this.name + "; вік: " + _age);
   };
   this.getUserAge = function() {
      return _age;
   }
   this.setUserAge = function(age) {
      if (typeof age === "number" && age> 0 && age <110) {
         _age = age;
      } else {
         console.log("Невідповідне значення");
      }
   }
}
var george = new User("George");
console.log(george._age); // undefined - _age - локальна змінна
console.log(george.getUserAge()); // 1
george.setUserAge(32);
console.log(george.getUserAge()); // 32
george.setUserAge("54"); // Невідповідне значення
george.setUserAge(123); // Невідповідне значення
```

У конструкторі *User* оголошується локальна змінна *_age* замість властивості *age*. Як правило, назви локальних змінних в конструкторах починаються зі знака підкреслення.

Для того, щоб працювати з віком користувача ззовні, визначаються два методи. Метод *getUserAge()* призначений для отримання значення змінної *_age*. Цей метод ще називають **гетер (getter)**. Другий метод – *setUserAge*, який ще називається **сетер (setter)**, призначений для задання значення змінної *_age*.

Перевагою такого підходу є більший контроль доступу зміни значення *_age*. Наприклад, можна використати супутні умови для перевірки типу значення (в даному випадку, тип повинен бути числовим) та самого значення (вік не може бути менше 0 і більше 110).

5.3.5. Успадкування

JavaScript підтримує успадкування. Це дозволяє при створенні нових типів об'єктів за необхідності успадкувати їх функціонал від уже існуючих. Наприклад, є **об'єкт *User***, що представляє користувача. Також є **об'єкт *Employee***, який представляє працівника. Але працівник також може бути користувачем і тому повинен мати всі його властивості та методи. Наприклад:

```
function User(name, age) { // конструктор користувача
   this.name = name;
   this.age = age;
   this.go = function() {
     document.write(this.name + " йде <br/>");
   }
   this.printInfo = function() {
     document.write( "Ім'я: "+this.name+ "; вік: " + this.age);
   };
}
User.prototype.MaxAge = 110;
   //конструктор працівника
function Employee(name, age, comp) {
   User.call(this, name, age);
   this.company = comp;
   this.printInfo = function() {
```

```
        document.write("Ім'я: "+this.name+"; вік: "+this.age+ ";
        компанія: " + this.company);
    };
}
Employee.prototype = Object.create(User.prototype);
var george = new User("George", 26);
var yuzik = new Employee("Yuzik", 32, "Google");
george.go();
yuzik.go();
george.printInfo();
yuzik.printInfo();
console.log(yuzik.MaxAge); //110
```

Отже, спочатку визначається конструктор *User* і до його прототипу додається властивість *MaxAge*. Потім визначається тип *Employee*. У конструкторі *Employee* відбувається виклик конструктора *User* наступним чином:

User.call(this, name, age);

Передача першого параметра дозволяє викликати конструктор *User* для об'єкта, що створюється конструктором *Employee*. Завдяки цьому всі властивості і методи, визначені в конструкторі *User*, також переходять на об'єкт *Employee*. Крім того, необхідно успадкувати також і прототип *User*. Для цього призначений виклик:

Employee.prototype = Object.create(User.prototype);

Метод *Object.create()* дозволяє створити об'єкт прототипу *User* для прототипу *Employee*. При цьому за необхідності прототип *Employee* можна доповнити новими властивостями і методами.

При успадкуванні можна перевизначити успадкований функціонал. Наприклад, для *Employee* перевизначено метод *printInfo()*, успадкований від *User*, доповнивши виведення цього методу новою властивістю *company*.

5.3.6. Класи

Із введенням стандарту ES2015 (ES6) в JavaScript з'явився новий спосіб визначення об'єктів – за допомогою класів. **Клас** являє собою опис об'єкта, його стану і поведінки, а **об'єкт** є конкретним екземпляром класу.

Для визначення класу використовується **ключове слово *class***:

```
class Person {
}
```

Після слова *class* вказується **назва класу** (в даному випадку, клас називається *Person*), потім у фігурних дужках визначається **тіло класу**. Можна визначити й **анонімний клас** і присвоїти його змінній:

```
let Person = class {}
```

Після цього можна **створити об'єкти класу** за допомогою конструктора:

```
let alex = new Person();
let ivo = new Person();
```

Для створення об'єкта за допомогою конструктора використовується ключове слово *new*, після якого виклик конструктора, фактично, виклик функції по імені класу. За замовчуванням, класи мають один **конструктор без параметрів**. При виклику такого конструктора в нього не передається жодних параметрів. Також можна визначити в класі свої конструктори, властивості і методи:

```
class Person {
   constructor(n, a) {
      this.name = n;
      this.age = a;
   }
   print() {
      console.log(this.name, this.age);
   }
}
let alex = new Person("Alex", 34);
alex.print(); // Alex 34
console.log(alex.name); // Alex
```

Конструктор визначається за допомогою методу з іменем *constructor*. По суті, це звичайний метод, який може мати параметри. Основна мета конструктора – ініціалізувати об'єкт початковими даними. В прикладі вище в конструктор передаються два значення – для імені і віку користувача, щоб ініціалізувати ними властивості об'єкта: *new Person("Alex", 34)*.

Для зберігання стану в класі призначені **властивості**. Для їх визначення використовується ключове слово *this*. У даному випадку, в класі дві властивості: *name* і *age*.

Поведінку класу визначають **методи**. У прикладі вище визначено метод *print()*, який виводить значення властивостей на консоль.

Одні класи можуть успадковуватися від інших. **Спадкування** дозволяє скоротити обсяг коду в класах-нащадках. Для успадкування одного класу від іншого у визначенні класу застосовується **ключове слово *extends***, після якого вказується назва базового класу.

Наприклад, визначимо класи:

```
class Person {
    constructor(n, a) {
        this.name = n;
        this.age = a;
    }
    print() {
        console.log(this.name, this.age);
    }
}
class Employee extends Person {
    constructor(n, a, c) {
        super(n, a);
        this.company = c;
    }
    print() {
        super.print();
        console.log("Працює у", this.company);
    }
    work() {
        console.log(this.name, " є працьовитим");
    }
}
let alex = new Person("Alex", 34);
let george = new Employee("George", 36, "Google");
alex.print(); // Alex 34
george.print(); // George 36 Працює у Google
```

george.work(); // George є працьовитим

У цьому випадку, клас *Employee* успадковується від класу *Person*. Клас *Person* ще називається **базовим класом, класом-батьком, суперкласом**, а клас *Employee* – **класом-спадкоємцем, підкласом, похідним, дочірнім класом**.

У похідному класі, як і базовому, можуть визначатися конструктори, властивості, методи. Разом з тим, за допомогою **ключового слова *super*** похідний клас може посилатися на функціонал, визначений у базовому. Наприклад, у конструкторі *Employee* рекомендується не задавати значення успадкованих властивостей *name* і *age*, а за допомогою виразу *super (n, a);* **викликати конструктор базового класу** і таким чином передати роботу по заданню цих властивостей базовому класу. Аналогічно, в методі *print* у класі *Employee* через виклик *super.print()* відбувається звернення до реалізації методу *print()* класу *Person*.

Консольний вивід програми:

Alex 34
George 36
Працює у Google
George є працьовитим

Статичні методи викликаються для всього класу в цілому, а не для окремого об'єкту. Для їх визначення застосовується **оператор *static***. Наприклад:

```
static nameToUpper(person) {
      return person.name.toUpperCase();
   }
```

Виклик статичного методу:

```
let jerry = new Person("Jerry", 39);
let jerryName = Person.nameToUpper(jerry);
console.log(jerryName);       // JERRY
```

У цьому випадку, визначено статичний метод *nameToUpper()*, який як параметр приймає об'єкт *person* і перетворює його ім'я у верхній регістр. Оскільки статичний метод відноситься до класу в цілому, а не до об'єкта, то не можна використовувати в ньому ключове слово *this*, тобто через *this* звертатися до властивостей об'єкта.

5.4. Об’єкти Date, Math, Number

Об’єкт Date дозволяє працювати з датами і часом в JavaScript. Існують різні способи створення об’єкта Date. **Перший спосіб** полягає у використанні порожнього конструктора без параметрів:

var currentDate = new Date();

У цьому випадку, об’єкт буде містити поточну дату комп’ютера.

Другий спосіб полягає у передачі в конструктор Date кількості мілісекунд, які пройшли з початку епохи Unix, тобто з 1 січня 1970 року 00:00:00 GMT:

var myDate = new Date(1359270000000);

Третій спосіб полягає в передачі в конструктор Date дня, місяця і року дати:

var myDate1= new Date("27 March 2008");

// або так

var myDate2 = new Date("3/27/2008");

Якщо використовується повна назва місяця, то вона пишеться англійською; якщо використовується скорочений варіант, то формат *“місяць/день/рік”*.

Четвертий спосіб полягає в передачі в конструктор Date всіх параметрів дати і часу:

// Tue Dec 25 2012 18:30:20 GMT + 0300 (RTZ 2 (зима))

var myDate = new Date(2012, 11, 25, 18, 30, 20, 10);

У цьому випадку, в зазначеному порядку використовуються наступні параметри:

new Date (рік, місяць, число, час, хвилини, секунди, мілісекунди).

При цьому потрібно враховувати, що відлік місяців починається з нуля, тобто січень – 0, а грудень – 11.

Для отримання різних складових дати використовуються методи:

- *getDate()* повертає день місяця;
- *getDay()* повертає день тижня (відлік починається з 0 – неділя, останній день – 6 – субота);
- *getMonth()* повертає номер місяця (відлік починається з

нуля, тобто місяць з номером 0 – січень);

- *getFullYear()* повертає рік;
- *toDateString()* повертає повну дату у вигляді рядка;
- *getHours()* повертає годину (від 0 до 23);
- *getMinutes()* повертає хвилини (від 0 до 59);
- *getSeconds()* повертає секунди (від 0 до 59);
- *getMilliseconds()* повертає мілісекунди (від 0 до 999);
- *toTimeString()* повертає повний час у вигляді рядка.

Для задання дати використовують такі методи об'єкта Date:

- *setDate()* задає день дати;
- *setMonth()* задає місяць (відлік починається з нуля, тобто місяць з номером 0 - січень);
- *setFullYear()* задає рік;
- *setHours()* задає годину;
- *setMinutes()* задає кількість хвилин;
- *setSeconds()* задає кількість секунд;
- *setMilliseconds()*задає кількість мілісекунд.

Об'єкт Math надає набір математичних функцій, які можна використовувати при обчисленнях. Розглянемо основні з них:

- *abs()* повертає абсолютне значення числа;
- *min()* і *max()* повертають, відповідно, мінімальне та максимальне значення з набору чисел (ці функції не обов'язково повинні приймати два числа, в них можна передавати більшу кількість чисел);
- *ceil()* округлює число до найближчого більшого цілого числа (фактично, повертає цілу частину числа +1);
- *floor()* округлює число до найближчого меншого цілого числа (фактично, повертає цілу частину числа);
- *round()* заокруглює число до найближчого меншого цілого числа, якщо його десяткова частина менше 0.5; якщо ж десяткова частина дорівнює або більше 0.5, то округлення до найближчого більшого цілого числа;
- *random()* повертає випадкове число з плаваючою крапкою з діапазону від 0 до 1;
- *pow()* повертає число, піднесене до степеня;

- *sqrt()* повертає квадратний корінь числа;
- *log()* повертає натуральний (за основою *e=2,718*) логарифм числа;
- тригонометричні функції *sin(), cos(), tan()* повертають, відповідно, синус, косинус, тангенс кута, що задається параметром у радіанах;
- обернені тригонометричні функції: *asin()* обчислює арксинус числа, *acos*() - арккосинус, а *atan()* - арктангенс.

Крім методів, об'єкт Math, також визначає **набір вбудованих констант**, які можна використовувати в різних обчисленнях, зокрема:

- Math.PI (число PI): 3.141592653589793;
- Math.E (число *e* або число Ейлера): 2.718281828459045.

Об'єкт Number представляє число. Щоб створити число, потрібно передати в конструктор Number число або рядок, що містить число:

```
var number1 = new Number(34);
var number2 = new Number('34');
document.write(number1 + number2); // 68
```

Визначення *number1* і *number2*, у даному випадку, практично рівносильні. Також можна створити об'єкт Number, присвоївши змінній число:

```
var z = 34;
```

Об'єкт Number надає ряд властивостей і методів. Деякі його властивості:

- *Number.MAX_VALUE*: найбільше можливе число, приблизно 1.79E + 308, числа, які більше цього значення, розглядаються як *Infinity*;
- *Number.MIN_VALUE*: найменше можливе додатнє число, приблизно 5e-324;
- *Number.NaN*: спеціальне значення, яке вказує, що об'єкт не є числом;
- *Number.NEGATIVE_INFINITY*: значення, яке позначає негативну нескінченність (*-Infinity*) і яке виникає при переповненні, наприклад, при додаванні двох від'ємних чисел, які за модулем рівні *Number.MAX_VALUE*;
- *Number.POSITIVE_INFINITY*: позитивна нескінченність

(*Infinity*), також, як і негативна, виникає при переповненні, тільки тепер в додатному напрямку.

Деякі основні методи:

- *isNaN()* визначає, чи є об'єкт числом, якщо об'єкт не є числом, то повертає значення *true*;
- *parseFloat()* перетворює рядок в число з плаваючою крапкою;
- *toFixed()* залишає в числі з плаваючою точкою певну кількість знаків у дробовій частині.

5.5. Робота з рядками та регулярними виразами

Для створення рядків можна присвоїти змінній рядок:

```
let name = "George";
```

Для роботи з рядками призначений **об'єкт String**, тому також можна використовувати конструктор String:

```
var name = new String("George");
```

Як правило, використовується перший спосіб, який є більш простим. У першому випадку, JavaScript при необхідності автоматично перетворює змінну примітивного типу в об'єкт String. Об'єкт String має великий набір властивостей і методів, за допомогою яких можна працювати з рядками:

1. Властивість length повертає довжину рядка:

```
var hello = "привіт світ";
console.log("У рядку '"+hello+"' "+hello.length+" символів");
   // У рядку 'привіт світ' 11 символів
```

2. Метод repeat() дозволяє створити рядок шляхом багаторазового повторення іншого рядка, приймає як аргумент кількість повторів, причому повтори не розділяються жодними символами:

```
console.log(hello.repeat(2)); //привіт світпривіт світ
```

3. Шаблони рядків дозволяють вставляти в рядок значення змінних. Для цього рядок обгортають косими лапками:

```
let nameUser = "George";
let helloUser = `Hello ${nameUser}`;
console.log(helloUser); // Hello George
let ageUser = 23;
```

let infoUser = `${nameUser} is ${ageUser} years old`;
console.log(infoUser); // George is 23 years old

Для вставки значення змінної в рядок, змінна береться у фігурні дужки, перед якими ставиться знак долара. Також замість значень змінних в рядок можуть вставлятися значення властивостей складених об'єктів або результати виразів.

4. Для пошуку в рядку деякого підрядка використовуються методи *indexOf()* (індекс першого входження підрядка) і *lastIndexOf()* (індекс останнього входження підрядка). Ці методи мають два параметри:

- підрядок, який потрібно знайти;
- необов'язковий параметр, який вказує, з якого символу виконувати пошук підрядка в рядку.

Обидва ці методи повертають індекс символу, з якого в рядку починається підрядок. Якщо підрядок не знайдено, то повертається число «*-1*».

5. Ще один метод includes() повертає *true*, якщо рядок містить певний підрядок.

6. Для того, щоб отримати (скопіювати) з рядка підрядок, застосовуються методи *substr()* і *substring()*.

Метод *substring()* дозволяє отримати (скопіювати) символи з рядка (підрядок) між двома заданими індексами, які передаються як параметри методу, або від певного індексу до кінця рядка, якщо другий параметр не вказано.

Метод *substr()* також дозволяє отримати (скопіювати) з рядка певну кількість символів (підрядок), починаючи із заданого індексу. Цей метод приймає як перший параметр початковий індекс підрядка, а як другий – довжину підрядка, який потрібно отримати. Якщо другий параметр не вказано, то копіюється решта рядка.

7. Для зміни регістру призначені методи *toLowerCase()* (перетворення в нижній регістр) і *toUpperCase()* (перетворення у верхній регістр).

8. Щоб отримати певний символ в рядку за індексом, застосовують методи *charAt()* і *charCodeAt()*. Обидва методи як параметр приймають індекс символу: але як результат метод *charAt()* повертає сам символ, а метод *charCodeAt()* – числовий

код цього символу.

9. Для видалення початкових і кінцевих пробілів у рядку, використовується метод *trim()*.

10. Метод *concat()* об'єднує два рядки:

```
let hello = "Привіт ";
let world = "світ";
hello = hello.concat(world);
console.log(hello); // Привіт світ
```

11. Метод *replace()* замінює перше входження одного підрядка на інший:

```
let hello = "Добрий день";
hello = hello.replace("день", "вечір");
console.log(hello); // Добрий вечір
```

Перший параметр методу вказує, який підрядок потрібно замінити, а другий – на який підрядок замінити.

12. Метод *split()* розбиває рядок на масив підрядків за певним роздільником. Як роздільник використовується параметр, що передається в метод:

```
var message = "Сьогодні була чудова погода";
var strArray = message.split(" ");
for (var str in strArray)
   console.log(strArray[str]);
```

Виведення у браузері:

сьогодні
була
чудова
погода

13. Метод *startsWith()* повертає *true*, якщо рядок починається з певного підрядка. Також **метод *endsWith()*** повертає *true*, якщо рядок закінчується на певний підрядок. При цьому має значення регістр символів.

Додатковий другий параметр дозволяє вказати індекс (для *startsWith* – індекс з початку, а для *endsWith* – індекс з кінця рядка), з якого буде здійснюватися порівняння.

Регулярний вираз являє собою шаблон, який використовується для пошуку або модифікації рядка. Для роботи з регулярними виразами в JavaScript визначено **об'єкт RegExp**. Визначити регулярний вираз можна двома способами:

var myExp = /hello/;
var myExp = new RegExp("hello");

Наведений регулярний вираз є найпростішим: складається з одного слова *"hello"*. У першому випадку вираз поміщається між двома косими слешами, а в другому – використовується конструктор *RegExp*, в який передається параметр у вигляді рядка.

Щоб визначити, **чи відповідає рядок регулярному виразу**, в об'єкті RegExp визначено **метод *test()***. Цей метод повертає ***true***, якщо рядок відповідає регулярному виразу, і ***false***, якщо не відповідає:

var Text = "hello world!";
var Exp = /hello/;
var res = ***Exp.test(Text);***
*document.write(res + "
"); // true*
Text = "nice day";
res = ***Exp.test(Text);***
document.write(res); // false - в рядку Text немає "hello"

Аналогічно працює **метод *exec*** – також перевіряє, чи відповідає рядок регулярному виразу, але повертає ту частину рядка, яка відповідає виразу. Якщо відповідностей немає, то повертає значення *null*.

Регулярний вираз не обов'язково складається зі звичайних рядків, він також може містити **спеціальні елементи синтаксису регулярних виразів**. Одним із таких елементів є **набори символів**, вкладені в квадратні дужки, наприклад:

var Text = "обороноздатність";
var Exp = /[абв]/;
var res = Exp.test(Text);
*document.write(res + "
"); // true*

Вираз *[абв]* вказує на те, що рядок повинен містити хоча б одну із цих трьох букв. Щоб визначити наявність у рядку буквених символів із певного діапазону, потрібно вказати цей діапазон:

var Exp = /[а-я]/;

У цьому випадку рядок повинен містити хоча б один символ з діапазону *а-я*. Якщо навпаки, потрібно, щоб рядок складався не тільки із символів вказаного діапазону, то в квадратних

дужках перед перерахуванням символів ставлять знак ^:

var Exp = /[^а-я]/;

Наприклад:

```
var Text1 = "абббвбава";
var Text2 = "абббвбва_г";
var Text3 = "місто";
var exp = /[^а-в]/;
document.write(exp.test(Text1) + "<br/>");
   //false (рядок складається тільки з символів діапазону)
document.write(exp.test(Text2) + "<br/>");
   // true (рядок містить не тільки символи з діапазону)
document.write(exp.test(Text3) + "<br/>");
   // true (рядок містить не тільки символи з діапазону)
```

При необхідності можна **утворити комбінації виразів**:

```
var Text = "вдома";
var Exp = /[дт]о[нм]/;
var res = Exp.test(Text);
document.write(res); // true
```

Вираз *[дт]о[нм]* означає підрядок, де перша літера з діапазону *[дт]*, друга – літера *"о"* і третя – з *[нм]*, напр. "том", "дон", "тон".

При визначенні регулярних виразів можна використовувати **необов'язкові атрибути (прапорці)**. У такому випадку формат визначення регулярного виразу наступний:

var exp = /pattern/flags;

або що рівносильно

var exp = new RegExp(pattern, flags);

де необов'язковим атрибутом *flags* може бути будь-який із прапорців ("*g*", "*i*", і "*m*") або їх комбінація.

Прапорець *"g"* дозволяє знайти всі підрядки, які відповідають регулярному виразу (за замовчуванням, при пошуку підрядків регулярний вираз вибирає перший підрядок, який йому відповідає, хоча в рядку може бути багато підрядків, які також відповідають виразу); ***"i"*** означає знайти підрядки, які відповідають регулярному виразу, незалежно від регістру символів у рядку; "***m***" дозволяє знайти підрядки, які відповідають регулярному виразу, в багаторядковому тексті.

Кожна з властивостей *global, ignoreCase, multiline* має тип *boolean* та повертає *true*, якщо при визначенні регулярного виразу використовувався прапорець *"g"*, *"i"*, *"m"*, відповідно, в іншому випадку – *false*.

Властивість *global* повертає *true*, якщо при визначенні регулярного виразу використовувався прапорець *"g"*; в іншому випадку – *false*.

Властивість *ignoreCase* повертає *true*, якщо при визначенні регулярного виразу використовувався прапорець *"i"*; в іншому випадку – *false*.

Властивість *multiline* повертає *true*, якщо при визначенні регулярного виразу використовувався прапорець *"m"*, *false* – в протилежному випадку.

Наприклад:

```
var Text = "Привіт Світ";
var exp = /світ/i;
   //або що рівносильно var exp = new RegExp('світ', 'i');
var res = exp.test(Text);
document.write(res+ "<br/>"); // true
document.write(exp.ignoreCase+ "<br/>"); // true
document.write(exp.global); // false
```

Метод *match()* застосовується для пошуку всіх відповідностей в рядку:

```
var Text = "Він прийшов додому і зробив домашню роботу";
var Exp = /до[а-я]*/gi;
var res = Text.match(Exp);
res.forEach(function(value, index, array) {
   document.write(value + "<br/>");
})
```

Символ зірочка вказує на можливість наявності після рядка *"до"* довільної кількості символів з діапазону від *а* до *я*. У підсумку, в масиві *result* будуть слова: *додому*, *домашню*.

Метод *search* знаходить індекс першого входження відповідності в рядку:

```
var Text = "hello world";
var Exp = /wor/;
var res = Text.search(Exp);
document.write(res); // 6
```

Метод *replace* дозволяє замінити всі відповідності регулярного виразу певним рядком:

var menu = "Сніданок: каша, чай. Обід: суп, чай. Вечеря: салат, чай";

var Exp = /чай/gi;

menu = menu.replace(Exp, "кава");

document.write(menu);

Вивід у браузері:

Сніданок: каша, кава. Обід: суп, кава. Вечеря: салат, кава.

Регулярні вирази також можуть використовувати **метасимволи** – символи, які мають особливий зміст:

1) **\d** означає будь-яку цифру від 0 до 9;

2) **\D** означає довільний символ, який не є цифрою;

3) **\w** означає довільну літеру, цифру або символ підкреслення (діапазони: A-Z, a-z, 0-9, _);

4) **\W** відповідає будь-якому символу, який не є літерою, цифрою або символом підкреслення (тобто не знаходиться в діапазонах: A-Z, a-z, 0-9, _);

5) **\s** відповідає пробілу;

6) **\S** відповідає довільному символу, який не є пробілом;

7) **.** означає довільний символ.

Зауважимо, що метасимвол **\w** застосовується тільки для літер латинського алфавіту, він не охоплює кириличні символи.

Напр., стандартний формат номера телефону *+1-234-567-8901* відповідає регулярному виразу *\d-\d\d\d-\d\d\d-\d\d\d\d*. Замінимо числа номера нулями:

var Number = "+1-234-567-8901";

var Exp = /\d-\d\d\d-\d\d\d-\d\d\d\d/;

Number = Number.replace(Exp, "00000000000");

document.write(Number);

Крім вищерозглянутих елементів регулярних виразів, є ще одна група комбінацій, яка вказує, **як символи в рядку будуть повторюватися**. Такі комбінації ще називають **модифікаторами**:

1) ***{n}*** відповідає *n*-й кількості повторень попереднього символу, наприклад, *h{3}* відповідає підрядку *"hhh"*;

2) ***{n,}*** відповідає *n* і більшій кількості повторень попереднього символу, наприклад, *h{3,}* відповідає підрядкам

"hhh", *"hhhh"*, *"hhhhh"* і т.д.;

3) ***{n,m}*** відповідає від *n* до *m* повторень попереднього символу, наприклад, *h{2, 4}* відповідає підрядкам *"hh"*, *"hhh"*, *"hhhh"*;

4) ***?*** відповідає одному входженню попереднього символу в підрядок або його відсутності в підрядку, наприклад, */h?ome/* відповідає підрядкам *"home"* і *"ome"*;

5) **+** відповідає одному і більше повторень попереднього символу;

6) ***** відповідає будь-якій кількості повторень або відсутності попереднього символу;

7) ^ відповідає початку рядка, наприклад, *^h* відповідає рядку *"home"*, але не *"ohma"*, тому що *h* повинен бути початком рядка;

8) **$** відповідає кінцю рядка, наприклад, *к$* відповідає рядку *"будинок"*, тому що рядок повинен закінчуватися на букву *к*.

Наприклад, візьмемо той самий номер телефону. Йому відповідає формальний вираз \d-\d\d\d-\d\d\d-\d\d\d\d. Однак за допомогою вищерозглянутих комбінацій його можна спростити: \d-\d{3}-\d{3}-\d{4}.

Зазначимо, що оскільки **символи ?, +, *** мають особливий зміст у регулярних виразах, то щоб їх використовувати у звичайному розумінні їхнього значення (наприклад, поставити знак плюс в номері телефону), потрібно їх екранувати за допомогою слеша: \+\d-\d{3}-\d{3}-\d{4}.

Розглянемо застосування **комбінації** '**\b**', яка вказує на відповідність в межах слова. Наприклад, є рядок: *"Мови навчання: Java, JavaScript, C++"*. Припустимо, що в рядку потрібно замінити слово *"Java"* на *"C#"*. Але проста заміна приведе також до заміни рядка "JavaScript" на "C#Script", що неприпустимо. В цьому випадку можна виконати заміну, якщо регулярний вираз відповідає всьому слову:

```
var Text = "Мови навчання: Java, JavaScript, C++";
var Exp = /Java\b/g;
var result = Text.replace(Exp, "C#");
document.write(result);
   // Мови навчання: C#, JavaScript, C++
```

При використанні '\b' потрібно враховувати, що в JavaScript відсутня повноцінна підтримка Unicode, тому '\b' може застосовуватися тільки до англомовних слів.

5.6. Об’єктна модель браузера (BOM)

Велике значення в JavaScript має робота з веб-браузером і тими об’єктами, які він надає. Наприклад, використання об’єктів браузера дозволяє маніпулювати елементами html, які є на сторінці, або взаємодіяти з користувачем. Усі об’єкти, через які JavaScript взаємодіє з браузером, описуються поняттям **Browser Object Model** (об’єктна модель браузера). Browser Object Model можна зобразити у вигляді схеми (рис. 5.1).

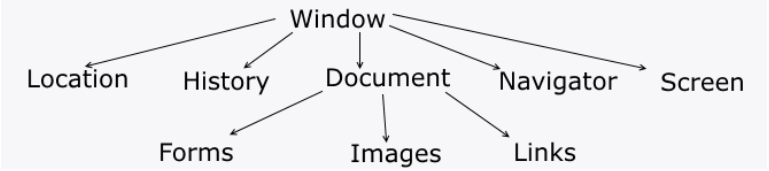


Рис. 5.1. Об’єктна модель браузера

На вершині знаходиться головний об’єкт – **об’єкт window**, який представляє браузер. Цей об’єкт, в свою чергу, включає ряд інших об’єктів, зокрема, **об’єкт document**, який представляє окрему веб-сторінку, що відображається в браузері.

Об’єкт window представляє вікно веб-браузера, в якому розміщуються веб-сторінки. window є глобальним об’єктом, тому при доступі до його властивостей і методів необов’язково використовувати його ім’я. Наприклад, window має **метод *alert()***, який відображає діалогове вікно з повідомленням:

window.alert("Привіт світ!");

Але window можна опускати, тобто

alert("Привіт світ!");

Для взаємодії з користувачем в об’єкті window визначено ряд методів, які дозволяють створювати діалогові вікна:

1. Метод *alert()* виводить діалогове вікно з повідомленням.

2. Метод *confirm()* відображає діалогове вікно з

повідомленням, в якому користувач повинен підтвердити дію, наявні дві кнопки: *OK* і *Скасувати*. Залежно від вибору користувача, метод повертає *true* (якщо користувач натиснув *OK*) або *false* (якщо користувач натиснув *Скасувати*):

```
var result = confirm ( "Завершити виконання програми?");
if (result === true)
   document.write ( "Робота програми завершена!");
else
   document.write ( "Програма продовжує працювати!");
```

3. Метод *prompt()* дозволяє за допомогою діалогового вікна запитувати у користувача будь-які дані. Цей метод повертає введене користувачем значення:

```
var age = prompt ( "Введіть свій вік:");
document.write ( "Вам " + age + " років");
```

Якщо користувач відмовиться вводити значення і натисне на кнопку *Скасувати*, то метод поверне значення *null*.

Об’єкт window також надає ряд методів для управління вікнами браузера:

1. Метод *open()* відкриває вказаний ресурс у новому вікні браузера:

```
var popup = window.open ('https://microsoft.com', 'Microsoft',
                'width = 400, height = 400, resizable = yes');
```

Метод *open()* має набір параметрів: шлях до ресурсу, описову назву для вікна і третій параметр – набір стильових значень вікна (ширина, висота,…). Метод повертає посилання на об’єкт нового вікна.

2. За допомогою **методу *close()*** можна закрити вікно.

3. Метод *moveTo()* дозволяє перемістити вікно на нову позицію.

4. Метод *resizeTo()* дозволяє змінити розміри вікна:

```
var popup = window.open ( 'https://microsoft.com', 'Microsoft',
'width = 400, height = 400, resizable = yes');
popup.resizeTo (500,350); // 500 - ширина і 350 – висота
```

Об’єкт history призначений для зберігання історії відвідувань веб-сторінок у браузері. Цей об’єкт доступний через об’єкт window. Всі відомості про відвідування користувача зберігаються в спеціальному стеку (history stack). За допомогою

властивості *length* можна дізнатися, скільки веб-сторінок зберігається в стеку:

document.write ("В історії " + history.length + " сторінок");

Для переходу між сторінками історії в об'єкті history визначені **методи: *back()*** (перехід до попередньої переглянутої сторінки) і ***forward()*** (перейти до наступної переглянутої сторінки):

history.back();

Також в об'єкті history визначений спеціальний **метод *go()***, який дозволяє переміститись вперед або назад в історії на певну кількість сторінок. Для переміщення назад в метод передається від'ємне значення, для переміщеня вперед – додатне:

history.go(-3);

history.go(3);

Об'єкт location містить інформацію про розташування поточної веб-сторінки: URL, інформацію про сервер, номер порту, протокол. За допомогою властивостей об'єкта location можна отримати цю інформацію:

- *href*: повний рядок запиту до ресурсу;
- *pathname*: шлях до ресурсу;
- *origin*: загальна схема запиту;
- *protocol*: протокол;
- *port*: порт, який використовується ресурсом;
- *host*: хост;
- *hostname*: назва хосту;
- *hash*: якщо рядок запиту містить символ решітки (#), то ця властивість повертає ту частину рядка, яка після цього символу;
- *search*: якщо рядок запиту містить знак питання (?), то ця властивість повертає ту частину рядка, яка після знака питання.

Об'єкт navigator містить інформацію про браузер і операційну систему, в якій браузер запущено. Він визначає ряд властивостей і методів, основною з яких є **властивість *userAgent***, що представляє браузер користувача:

document.write (navigator.userAgent);

Ця властивість зберігає повний рядок юзер-агента, наприклад: "Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.69

Safari/537.36". Щоб виокремити з цієї інформації безпосередньо браузер, потрібно знайти в ній назву браузера.

Для виконання дій через певні проміжки часу в об'єкті window передбачені **функції таймерів**. Є два типи таймерів: одні виконуються тільки один раз, а інші періодично через проміжок часу. **Для одноразового виконання дій через проміжок часу** призначена **функція *setTimeout()***, яка має два параметри:

var timerId = setTimeout (someFunction, period)

Параметр *period* визначає часовий проміжок, через який буде виконуватися функція *someFunction*. Як результат функція повертає *id* таймера.

```
function timerFunction () {
   document.write ( "виконання функції setTimeout");
}
setTimeout (timerFunction, 3000);
```

У цьому випадку через 3 секунди після завантаження сторінки відбудеться спрацьовування функції *timerFunction*. Щоб зупинити таймер, застосовується **функція *clearTimeout()***.

Функції *setInterval()* і *clearInterval()* працюють, аналогічно функціям *setTimeout()* і *clearTimeout()*, з відмінністю, що *setInterval()* періодично виконує певну функцію через проміжок часу. Наприклад, веб-сторінка з виведенням поточного часу:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <title>Поточний час</title>
</head>
<body>
  <div id="time"></div>
  <script>
      function updateTime() {
       document.getElementById("time").innerHTML =
          new Date().toTimeString();
      }
      setInterval (updateTime, 1000);
</script>
```

</body>
</html>

В результаті, через кожну секунду (1000 мілісекунд) викликається функція *updateTime()*, яка оновлює вміст поля * *, встановлюючи його html-кодом поточний час.

5.7. Об'єктна модель документа (DOM)

Одним з ключових завдань JavaScript є взаємодія з користувачем і маніпуляція елементами веб-сторінки. Для JavaScript веб-сторінка доступна у вигляді **об'єктної моделі документа (Document Object Model)** або скорочено **DOM**. DOM описує структуру веб-сторінки у вигляді деревоподібного представлення і надає розробникам можливість отримати доступ до окремих елементів веб-сторінки.

Важливо не плутати поняття BOM (Browser Object Model – об'єктна модель браузера) і DOM (об'єктна модель документа). Якщо **BOM** надає доступ до браузера і його властивостей в цілому, то **DOM** надає доступ до окремої веб-сторінки або html-документа і його елементів.

Наприклад, розглянемо найпростішу сторінку:

```
<!DOCTYPE html>
<html>
<head>
   <title> Заголовок сторінки </title>
</head>
<body>
   <h2> Page Header </h2>
   <div>
      <h3> Block Header </h3>
      <p> Text </p>
   </div>
</body>
</html>
```

Дерево DOM для цієї сторінки має такий вигляд (рис. 5.2):

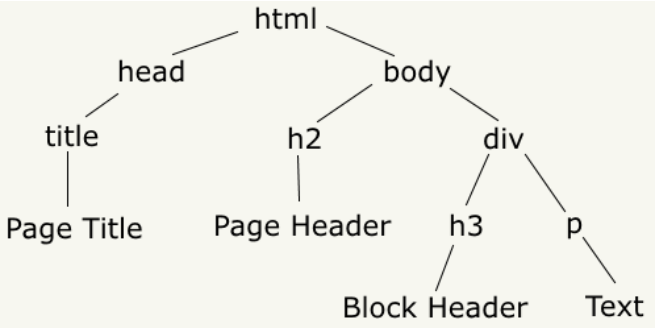


Рис. 5.2. Дерево DOM

Таким чином, всі компоненти впорядковані в DOM в ієрархічному порядку, де **кожен компонент являє собою окремий вузол**. Тобто кожний елемент, наприклад елемент *div*, є вузлом. Але і текст всередині елемента також є окремим вузлом.

Існують такі **види вузлів**:

1. **Element**: html-елемент.
2. **Attr**: атрибут html-елемента.
3. **Text**: текст елемента.
4. **CDATASection**: секція CDATA в документі XML.
5. **EntityReference**: посилання на сутність XML-документа.
6. **Entity**: необроблена сутність DTD.
7. **ProcessingInstruction**: інструкція обробки веб-сторінки.
8. **Comment**: елемент коментаря.
9. **Document**: кореневий вузол html-документа.
10. **DocumentType**: DTD або встановлений режим роботи XML-документа.
11. **DocumentFragment**: місце для тимчасового зберігання частин документа.
12. **Notation**: нотація, оголошена в DTD.

Незважаючи на таку велику кількість типів вузлів, як правило, робота здійснюється тільки з деякими з них.

Для роботи зі структурою DOM в JavaScript призначений **об'єкт document**, який визначений в глобальному об'єкті window. Об'єкт document надає набір властивостей і методів для управління елементами сторінки.

Для пошуку елементів на сторінці використовуться такі методи:

- *getElementById(value)* вибирає елемент, у якого **атрибут *id*** дорівнює *value*;
- *getElementsByTagName(value)* вибирає всі елементи, у яких **тег** дорівнює *value*;
- *getElementsByClassName(value)* вибирає всі елементи, які мають **клас** *value*;
- *querySelector(value)* вибирає перший елемент, який відповідає **css-селектору** *value*;
- *querySelectorAll(value)* вибирає всі елементи, які відповідають **css-селектору** *value*.

Наприклад, знайдемо елемент по *id*:

```
…
  <h3 id = "header"> Блок хедеру</h3>
…
<script>
  var header = document.getElementById("header");
  document.write("Текст заголовка: " + header.innerText);
</script>
```

За допомогою виклику методу

document.getElementById("header")

знаходимо елемент, у якого *id = "header"*, і за допомогою властивості *innerText* отримуємо текст знайденого елемента.

Крім раніше розглянутих методів, **властивості об'єкта document** дозволяють звернутися до певних елементів веб-сторінки:

- *documentElement* надає доступ до кореневого елемента *<html>*;
- *body* надає доступ до елемента *<body>* на веб-сторінці;
- *images* містить колекцію всіх об'єктів зображень (елементів *img*);
- *links* містить колекцію посилань – елементів *<a>* і *<area>*, у яких визначено атрибут *href*;
- *anchors* надає доступ до колекції елементів *<a>*, у яких визначено атрибут *name*;
- *forms* містить колекцію всіх форм на веб-сторінці.

Ці властивості не надають доступ до всіх елементів, однак дозволяють отримати найбільш часто використовувані елементи на веб-сторінці. Наприклад, отримаємо всі зображення на сторінці:

```
…
   <img src = "picture1.png" alt = "Картинка 1" />
   <img src = "picture2.png" alt = "Картинка 2" />
   <img src = "picture3.png" alt = "Картинка 3" />
<script>
 var allImages = document.images;
   // змінимо перше зображення
 allImages[0].src = "pictures/picture4.png";
 allImages[0].alt = "Нова картинка";
   // перебір всіх зображень
 for (var i = 0; i < allImages.length; i++) {
  document.write("<br/>" + allImages[i].src);
  document.write("<br/>" + allImages[i].alt);
}
</script>
…
```

Подібно до того, як в html-коді можна задати значення атрибутів для елемента *img*, так і в коді javascript можна через властивості *src* і *alt* отримати та задати значення цих атрибутів.

Кожен окремий вузол, будь то html-елемент, його атрибут або текст, в структурі DOM представлений **об'єктом Node**. Цей об'єкт **надає набір властивостей**, за допомогою яких можна отримати інформацію про даний вузол:

- *childNodes* містить колекцію дочірніх вузлів;
- *firstChild* повертає перший дочірній вузол поточного вузла;
- *lastChild* повертає останній дочірній вузол поточного вузла;
- *previousSibling* повертає попередній елемент, який знаходиться на одному рівні з поточним;
- *nextSibling* повертає наступний елемент, який знаходиться на одному рівні з поточним;
- *ownerDocument* повертає кореневий вузол документа;

• *parentNode* повертає елемент, який містить поточний вузол;

• *nodeName* повертає ім’я вузла;

• *nodeType* повертає тип вузла у вигляді числа (кожному типу відповідає певне число: 1 – елемент; 2 – атрибут; 3 – число);

• *nodeValue* повертає або встановлює значення вузла у вигляді простого тексту.

Розглянемо сторінку з наступним html-кодом:

```
<div class="article">
    <h2>Заголовок статті</h2>
    <p>Перший абзац</p>
    <p>Другий абзац</p>
</div>
```

Виконаємо для неї перебір усіх дочірніх вузлів елемента *div* із класом *article*:

```
<script>
  var artDiv = document.querySelector("div.article");
  var chNodes = artDiv.childNodes;
  for (var i = 0; i < chNodes.length; i++) {
    var type = "";
    if (chNodes[i].nodeType === 1)
       type = "елемент";
    else if (chNodes[i].nodeType === 2)
       type = "атрибут";
    else if (chNodes[i].nodeType === 3)
       type = "текст";
    console.log(chNodes[i].nodeName + ":" + type);
  }
</script>
```

Результат виведення у консолі:

```
#text:текст
H2:елемент
#text:текст
P:елемент
#text:текст
P:елемент
```

#text:текст

За допомогою методу *document.querySelector("div.article")* вибираємо елемент *div* з класом *article* і перебираємо його дочірні вузли. В циклі виводимо ім'я вузла та його тип за допомогою властивостей *nodeName* і *nodeType*. Кожному типу відповідає певне число.

Але незважаючи на те, що на сторінці в блоці *div* тільки три вузли: *h3* і два параграфи, консоль відобразить сім вузлів. Річ у тому, що пробіли між вузлами також вважаються окремими текстовими вузли. Якби пробілів не було:

* <h2>Заголовок статті</h2> Перший абзац Другий абзац *

то при переборі було б тільки три дочірні вузли, як і очікувалося, тобто

H2:елемент

P:елемент

P:елемент

Для створення елементів об'єкт document має такі методи:

1) *createElement(elementName)* створює елемент html, тег якого передається як параметр, повертає створений елемент;

2) *createTextNode(text)* створює і повертає текстовий вузол, як параметр передається текст вузла.

Наприклад:

```
var elem = document.createElement("div");
  //створюємо для елемента div текст
var elemText = document.createTextNode("Привіт світ");
```

Таким чином, змінна *elem* буде зберігати посилання на елемент *div*. Однак створити елемент недостатньо, його ще потрібно додати на веб-сторінку.

Для додавання елементів можна використовувати один з методів об'єкта Node:

1) *appendChild(newNode)* додає вузол *newNode* в кінець колекції дочірніх вузлів;

2) *insertBefore(newNode, referenceNode)* додає вузол *newNode* перед вузлом *referenceNode*.

Наприклад, додамо в наявний на сторінці блок *div*, заголовок *h2* із текстом *"Привіт світ"*:

```
<div class="article"> </div>
<script>
  var articleDiv = document.querySelector("div.article");
    // створюємо елемент
  var elem = document.createElement("h2");
    // створюємо для нього текст
  var elemText = document.createTextNode("Привіт світ");
    // додаємо в елемент текст як дочірній елемент
  elem.appendChild(elemText);
    // додаємо елемент в блок div
  articleDiv.appendChild(elem);
</script>
```

Елементи бувають досить складними за своєю структурою, і набагато простіше їх скопіювати, ніж створювати їх вміст. **Для копіювання вже наявних вузлів** у об'єкта Node можна використовувати **метод *cloneNode()***:

```
var articleDiv = document.querySelector( "div.article");
  // клонуємо елемент articleDiv
var newArticleDiv = articleDiv.cloneNode(true);
  // додаємо в кінець елемента body
document.body.appendChild (newArticleDiv);
```

У метод *cloneNode()* параметром передається логічне значення: *true* означає, що елемент буде копіюватися з усіма дочірніми вузлами; *false* – копіювання без дочірніх вузлів. Тобто, в даному випадку, копіюємо вузол з усім вмістом і додаємо його в кінець елемента *body*.

Для видалення елемента викликається метод *removeChild()* об'єкта Node. Цей метод видаляє один із дочірніх вузлів:

```
<div class="article">
      <h3>Заголовок </h3>
      <p>Перший параграф</p>
      <p>Другий параграф</p>
</div>
<script>
 var articleDiv = document.querySelector("div.article");
   /*знаходимо вузол, який будемо видаляти –
    перший параграф*/
```

```
 var removableNode=
    document.querySelectorAll("div.article p")[0];
  // видаляємо вузол
 articleDiv.removeChild(removableNode);
</script>
```

У цьому випадку видаляється перший параграф із блоку *div*.

Для заміни елемента застосовується метод *replaceChild(newNode, oldNode)* об'єкта Node. У цьому методі перший параметр – новий елемент, який замінює старий елемент *oldNode*, що передається другим параметром:

```
 <div class="article">
    <h3>Заголовок </h3>
    <p>Перший параграф</p>
    <p>Другий параграф</p>
 </div>
<script>
 var articleDiv = document.querySelector("div.article");
 /* знаходимо вузол, який будемо замінювати –
    перший параграф*/
 var oldNode = document.querySelectorAll("div.article p")[0];
 // створюємо елемент
 var newNode = document.createElement("h2");
 // створюємо для нього текст
 var elemText = document.createTextNode("Привіт світ");
 // додаємо текст в елемент як дочірній елемент
 newNode.appendChild(elemText);
 // замінюємо старий вузол новим
 articleDiv.replaceChild(newNode, oldNode);
</script>
```

Тут замінюємо перший параграф заголовком *h2*.

Крім методів і властивостей об'єкта Node, в JavaScript можна використовувати **властивості і методи об'єктів Element**. Важливо не плутати ці два об'єкти: Node і Element.

Node представляє всі вузли веб-сторінки, тоді як об'єкт **Element** – тільки html-елементи. Тобто об'єкти Element – це, фактично, ті ж самі вузли – об'єкти Node, у яких тип вузла (властивість *nodeType*) дорівнює 1.

Однією з ключових властивостей об'єкта Element є **властивість *tagName***, яка повертає тег елемента. Наприклад, отримаємо всі елементи, наявні на сторінці:

```
<script>
 function getChildren(element) {
  for (var index in element.childNodes) {
     if (element.childNodes[index].nodeType === 1) {
        console.log(element.childNodes[index].tagName);
        getChildren(element.childNodes[index]);
     }
  }
 }
 var root = document.documentElement;
 console.log(root.tagName);
 getChildren(root);
</script>
```

У прикладі вище спочатку отримується кореневий елемент *<html>* і потім за допомогою рекурсивної функції *getChildren* – всі вкладені елементи. Для отримання або задання текстового вмісту елемента можна використовувати властивість *innerText*, а для html-коду – властивість *innerHTML*:

```
var artDiv = document.querySelector("div.article");
console.log(artDiv.innerText);
console.log("----------------------------");
console.log(artDiv.innerHTML);
```

Зазначимо, що властивість *innerText* багато в чому аналогічна властивості *textContent*. Наприклад, такі виклики будуть рівносильні:

```
var pElem = document.querySelectorAll("div.article p")[0];
pElem.innerText = "hello";
pElem.textContent = "hello";
```

Задання html – коду для елемента:

```
var artDiv = document.querySelector("div.article");
artDiv.innerHTML="<h2>Hello World!!!</h2><p>text</p>";
```

Серед методів об'єкта Element варто відзначити **методи управління атрибутами**:

- *getAttribute(attr)* повертає значення атрибута *attr*;

• *setAttribute(attr,value)* задає для атрибуту *attr* значення *value*; якщо атрибута немає, то він додається;

• *removeAttribute(attr)* видаляє атрибут *attr* з його значенням.

Робота з атрибутами:

```
<div class="article">
    <h3>Заголовок </h3>
    <p>Перший параграф</p>
    <p>Другий параграф</p>
 </div>
<script>
  var artDiv = document.querySelector("div.article");
    // отримуємо атрибут style
  var styleVal = artDiv.getAttribute("style");
  console.log("До зміни атрибута:" + styleVal); //null
    // видаляємо атрибут
  artDiv.removeAttribute("style");
    // додаємо заново атрибут style
  artDiv.setAttribute("style", "color: blue;");
  styleVal = artDiv.getAttribute("style");
  console.log("Після зміни атрибута:"+styleVal);
    //color:blue;
</script>
```

Елементи мають **набір властивостей, які дозволяють** визначити **розмір елемента**. Але важливо розуміти різницю між усіма цими властивостями.

Властивості *offsetWidth* і *offsetHeight* визначають, відповідно, ширину і висоту елемента в пікселях. В ширину і висоту включаються границі елемента. Властивості *clientWidth* і *clientHeight* також визначають ширину і висоту елемента в пікселях, але вже без урахування границі.

Для визначення позиції елемента найбільш ефективним способом є метод *getBoundingClientRect()*. Цей метод повертає об'єкт з властивостями *top*, *bottom*, *left*, *right*, які вказують на зміщення елемента відносно верхнього лівого кута браузера.

Наприклад, розглянемо наступну сторінку:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Отримання розмірів елемента</title>
  <style>
   #rect { width: 100px; height: 100px;
           border: 3px solid black;
           background: #E4E6E1;
    }
  </style>
</head>
<body>
 <div id="rect"> </div>
 <script>
    var rectangle = document.getElementById("rect");
    var Rect = rectangle.getBoundingClientRect();
    console.log(" top:" + Rect.top);  // 8
    console.log(" bottom:" + Rect.bottom);  // 114=8+3+100+3
    console.log(" left:" + Rect.left);  // 8
    console.log(" right:" + Rect.right);    // 114=8+3+100+3
 </script>
</body>
</html>
```

Для роботи з властивостями стилів елементів у JavaScript застосовуються переважно два підходи:

- зміна властивості *style*;
- зміна значення атрибуту *class*.

Властивість *style* – складний об’єкт для управління стилем і безпосередньо зіставляється з атрибутом *style* html-елемента. Цей об’єкт містить набір властивостей CSS: *element.style.властивість*. Наприклад, задамо колір шрифту:

```
var root = document.documentElement;
  // задаємо стиль
root.style.color = "blue";
  // отримуємо значення стилю
document.write(root.style.color); // blue
```

У цьому випадку назва властивості *color* збігається з властивістю CSS. Аналогічно, можна встановити колір за допомогою CSS:

html { color: blue; }

Однак **ряд властивостей CSS в назвах мають дефіс**, наприклад *font-family*. В JavaScript для цих властивостей дефіс не вживається. Тому щоб їх використовувати потрібно першу літеру, яка йде після дефісу, записати у верхньому регістрі:

```
var root = document.documentElement;
root.style.fontFamily = "Verdana";
```

За допомогою властивості *className* можна задати атрибут *class* елемента html. Наприклад:

```
…
<style>
   .blueStyle{ font-family:Verdana; color:blue; }
   .article{ font-size:25px; }
</style>
</head>
<body>
  <div class="article">
     <h3>Заголовок статті</h3>
     <p>Перший абзац</p>
     <p>Другий абзац</p>
  </div>
  <script>
       var artDiv = document.querySelector("div.article");
         // задання  нового класу
       artDiv.className = "blueStyle";
         // отримуємо назву класу
      document.write(artDiv.className);
  </script>
</body>
</html>
```

Завдяки використанню класів не доведеться налаштовувати кожну окрему властивість CSS за допомогою властивості *style*. Але при цьому потрібно враховувати, що попереднє значення атрибута *class* видаляється. Тому, якщо потрібно доповнити клас, то треба об’єднати назву попереднього класу з назвою

нового класу, розділивши назви класів пробілом:

artDiv.className = artDiv.className+" "+"blueStyle";

Якщо потрібно повністю видалити всі класи, то можна присвоїти властивості порожній рядок:

artDiv.className = "";

Вище було розглянуто, як додавати класи до елемента. Однак **для управління множиною класів** набагато зручніше використовувати **властивість *classList***. Ця властивість є об'єктом, що реалізує такі методи:

- *add(className)* додає клас *className*;
- *remove(className)* видаляє клас *className*;
- *toggle(className)* перемикає для елемента клас на *className*: якщо класу немає, то він додається, якщо є, то видаляється.

Наприклад:

```
var articleDiv = document.querySelector("div.article");
  // видаляємо клас
articleDiv.classList.remove("article");
  // додаємо клас
articleDiv.classList.add("blueStyle");
  // перемикаємо клас
articleDiv.classList.toggle("article");
```

5.8. Події

Для взаємодії з користувачем в JavaScript визначено **механізм подій**. Наприклад, коли користувач натискає кнопку, то виникає подія натискання кнопки. **В JavaScript є такі типи подій:**

- події миші (переміщення курсору, натискання кнопки миші і т.д.);
- події клавіатури (натискання або відпускання клавіші клавіатури);
- події життєвого циклу елементів (наприклад, подія завантаження веб-сторінки);
- події елементів форм (натискання кнопки на формі, вибір елемента у випадаючому списку і т.д.);

- події, що виникають при зміні елементів DOM;
- події, що виникають при торканні на сенсорних екранах;
- події, що виникають при виникненні помилок.

У коді JavaScript можна визначити виникнення події і певним чином її обробити. Щоб обробити будь-яку подію, потрібно визначити для неї обробник. **Обробник події** – це блок коду (найчастіше, функція) на JavaScript, який виконується, як тільки відбулася подія. Саме завдяки механізму обробки подій, JavaScript-код може реагувати на дії користувача.

Існують **кілька способів призначити обробник події.** Розглянемо їх, починаючи з найпростішого:

1. Використання атрибуту html-елемента. Полягає в тому, що безпосередньо в html-розмітці присвоїти атрибуту *on<подія>* html-елемента обробник події. Тоді при настанні події виконуватиметься код, прописаний як значення атрибуту *on<подія>* елемента html.

Приклад найпростішої обробки події: на веб-сторінці є елемент *div* з **атрибутом *onclick***, що визначає обробник події натискання на блок *div*:

```
<div id = "rectangle" onclick = "alert('натискання')"
  style = "background-color: blue; width: 50px; height: 50px; ">
</div>
```

Отже, при натисканні кнопкою миші на блок *div* відбуватиметься вивід діалогового вікна з повіломленням "*натискання* ".

Можна **винести всі дії з обробки події в окрему функцію**:

```
<body>
  <div id="rect" onclick="displayMessage()"
    style="width:50px;height:50px; background-color:blue;">
  </div>
  <script>
     function displayMessage(){  alert('Натискання! '); }
  </script>
</body>
```

Тепер обробником події виступає функція *displayMessage()*.

В обробник можна передавати параметри. Наприклад, можна передати поточний об'єкт, через який виникає подія:

```
<a href="page1.html" onclick="return handler(this)">
        Сторінка 1
</a>
<script>
    function handler(obj) { alert(obj.href); return false; }
</script>
```

Ключове слово *this* вказує на поточний об'єкт – посилання, на яке натискає користувач. В коді обробника можна отримати цей об'єкт і звернутися до його властивостей, наприклад до властивості *href*.

Крім того, треба зазначити, що в цьому випадку обробник повертає результат. Хоча в першому прикладі з блоком *div* не було потрібно повернення результату. Річ у тому, що **для деяких обробників можна підтвердити або зупинити обробку події**. Наприклад, натискання на посилання повинно привести до переадресації, але повертаючи з обробника ***false***, можна зупинити стандартний шлях обробки події, і переадресації не відбудеться. Якщо ж повернути значення ***true***, то подія обробляється в стандартному порядку.

Якщо взагалі прибрати повернення результату, то подія буде оброблятися, так само як при поверненні значення *true*.

Крім безпосередньо елемента – джерела події, в обробник можна передавати **об'єкт *event***. Цей об'єкт не визначається розробником, це аргумент функції обробника, який зберігає всю інформацію про подію. Наприклад:

```
…
<div id = "rectangle" onclick = "handler(event)"> </div>
<script>
    function handler(e){ alert(e.type); } // отримуємо тип події
</script>
…
```

Тут за допомогою властивості *type* об'єкта *event* отримуємо тип події, в цьому випадку – тип *click*.

Це спосіб використання **вбудованих обробників** (inline event handler), які визначаються в коді елемента за допомогою атрибутів. Цей підхід чудово працює, але він має свої недоліки:

1. Код html змішується з кодом JavaScript, в зв'язку з чим стає важче розробляти, налагоджувати і підтримувати додаток.

2. Обробники подій можна задати тільки для вже створених на веб-сторінці елементів. Динамічно створювані елементи, в цьому випадку, позбавляються можливості обробки подій.

3. До елементу для однієї події може бути прикріплений тільки один обробник.

4. Не можна видалити обробник без зміни коду.

2. Використання властивостей DOM-об'єкта. Для вирішення проблем використання вбудованих обробників подій, застосовують властивості DOM-елементів.

Подібно до того, як у html-елементів є атрибути для призначення обробника, так і в коді javascript можна використовувати властивості елементів DOM вигляду *on<подія>*. Наприклад, використаємо властивість DOM-елемента *onclick*:

```
…
<div id = "rectangle"> </div>
…
<script>
    function handler(e) { alert(e.type);}
    document.getElementById("rectangle").onclick = handler;
</script>
…
```

Отже, достатньо присвоїти властивісті *onclick* функцію, яка використовується як обробник. За рахунок цього, html-код відділяється від коду javascript. Варто також зазначити, що в обробник події браузер автоматично передає об'єкт *event*, який зберігає всю інформацію про подію. Тому також можна отримати цей об'єкт у функції обробника як параметр.

Прибрати обробник можна, присвоївши йому значення *null*:

```
document.getElementById("rectangle").onclick = null;
```

Хоча властивості DOM-елементів вирішують ряд проблем, які пов'язані з використанням атрибутів html-елементів, але це також не оптимальний підхід. Зокрема, у елемента DOM може бути тільки одна властивість з одним і тим же іменем, напр. з іменем *onclick*. Тому призначити більше одного обробника також не вийде. Коли виникає така необхідність чи побажання? Наприклад, при кліку на кнопку потрібно, щоб одна частина коду підсвічувати її, а інша – видавала повідомлення.

3. Ще один спосіб визначення обробників подій представляє **використання слухача подій (event listeners)**. Слухач подій відслідковує подію, а обробник події – це код, який запускається як реакція на подію.

Для роботи зі слухачами подій в JavaScript є **об'єкт *EventTarget***, який визначає **методи: *addEventListener()*** для додавання слухача та ***removeEventListener()*** для видалення слухача. І оскільки html-елементи DOM теж є об'єктами *EventTarget*, то вони також мають ці методи. Фактично, слухачі виконують ті ж функції обробників.

Метод *addEventListener()* має два параметри: назва події без префікса *on* і функцію – обробник цієї події. Наприклад:

```
…
  <style>
    #rectangle{ width:100px; height:100px;
                background-color:red;
    }
  </style>
</head>
<body>
  <div id="rectangle"></div>
  <script>
    var rectangle = document.getElementById("rectangle");
    rectangle.addEventListener("click", function(e) {
      alert(e.type); });
  </script>
…
```

У цьому випадку обробляється подія *click*. Можна другим параметром передати назву функції:

```
<script>
  function handler(e) { alert (e.type); }
  var rectangle = document.getElementById("rectangle");
  rectangle.addEventListener("click", handler);
</script>
```

Видалення слухача аналогічно додаванню:

```
rectangle.removeEventListener("click", handler);
```

Перевагою використання слухачів подій є і те, що можна встановити для однієї події кілька функцій.

При обробці події браузер автоматично передає в функцію обробника як параметр **об'єкт *event***, який інкапсулює всю інформацію про подію. За допомогою **його властивостей** можна отримати таку інформацію:

- *bubbles* повертає *true*, якщо подія є висхідною, наприклад, якщо подія виникла на вкладеному елементі, то вона може бути оброблена на рівні батьківського елемента;
- *cancelable* повертає *true*, якщо можна відмінити стандартну обробку події;
- *currentTarget* визначає елемент, до якого прикріплений обробник події;
- *defaultPrevented* повертає *true*, якщо в об'єкта *event* був викликаний метод *preventDefault()*;
- *eventPhase* визначає стадію обробки події;
- *target* вказує на елемент, на якому було викликано подію;
- *timeStamp* зберігає час виникнення події;
- *clientX/clientY* повертає координати курсора в момент кліку відносно вікна для подій миші;
- *type* повертає тип події.

Наприклад:

```
…
<script>
  function handler(event) {
    console.log("Тип події:" + event.type);
    console.log(event.target);
  }
  var rectangle = document.getElementById("rectangle");
  rectangle.addEventListener("click", handler);
</script>
…
```

Результат у консольному вікні:

```
Тип події:click
<div id="rectangle"></div>
```

Причому, в цьому випадку, **властивість *target* є елементом**, тому можна маніпулювати ним, як і будь-яким іншим вузлом та елементом DOM. Наприклад, змінимо фоновий колір:

```
function handler(event) {
  event.target.style.backgroundColor = "blue";
}
```

За допомогою **методу *preventDefault()* об'єкта event** можна зупинити подальше виконання події. У ряді випадків цей метод не відіграє великої ролі. Однак в деяких ситуаціях він може бути корисним. Наприклад, при натисканні на посилання, можна за допомогою додаткової обробки визначити, чи треба переходити по посиланню, чи заборонити перехід. Або інший випадок: користувач відправляє дані форми, але під час обробки даних в обробнику події виявлено, що поля форми заповнені неправильно, тоді потрібно заборонити відправку.

При натисканні на елемент, на сторінці генерується подія натискання. Ця подія може поширюватися від елемента до елемента. Наприклад, при натисканні на блок *div*, також відбувається натискання і на елемент *body*, в якому блок *div* визначений. Тобто відбувається **поширення події**. **Є кілька форм поширення подій**:

- **висхідні**: подія поширюється вгору по дереву DOM від дочірніх вузлів до батьківських;
- **низхідні**: подія поширюється вниз по дереву DOM від батьківських вузлів до дочірніх, поки не досягне того елемента, на якому ця подія виникла.

Одні з найбільш часто використовуваних подій становлять **події миші:**

- *click* виникає при натисканні покажчиком миші на елемент;
- *mousedown* виникає при знаходженні покажчика миші на елементі, коли кнопка миші знаходиться в натиснутому стані;
- *mouseup* виникає при знаходженні покажчика миші на елементі під час відпускання кнопки миші;
- *mouseover* виникає при входженні покажчика миші в межі елемента;

• *mousemove* виникає при проходженні покажчика миші над елементом;

• *mouseout* виникає, коли покажчик миші виходить за межі елемента.

Об'єкт event є загальним для всіх подій. Однак для різних типів подій існують також свої об'єкти подій, які мають ряд своїх властивостей. Так, для роботи з подіями покажчика миші визначено **об'єкт MouseEvent**, який додає наступні **властивості**:

• *altKey* повертає *true*, якщо була натиснута клавіша *Alt* під час генерації події;

• *button* вказує, яка кнопка миші була натиснута;

• *clientX* повертає координату *X* вікна браузера, на якій знаходився покажчик миші під час генерації події;

• *clientY* повертає координату *Y* вікна браузера, на якій знаходився покажчик миші під час генерації події;

• *ctrlKey* повертає *true*, якщо була натиснута клавіша *Ctrl* під час генерації події;

• *metaKey* повертає *true*, якщо під час генерації події була натиснута метаклавіша клавіатури;

• *relatedTarget* визначає вторинне джерело виникнення події;

• *screenX* визначає координату *X* відносно верхнього лівого кута екрану монітора, на якій знаходився покажчик миші під час генерації події;

• *screenY* визначає координату *Y* відносно верхнього лівого кута екрану монітора, на якій знаходився покажчик миші під час генерації події;

• *shiftKey* повертає *true*, якщо була натиснута клавіша *Shift* під час генерації події.

Іншим поширеним типом подій є **події клавіатури**:

• *keydown* виникає при натисканні клавіші клавіатури і триває, поки клавіша натиснута;

• *keyup* виникає при відпуску клавіші клавіатури;

• *keypress* виникає при натисканні клавіші клавіатури, але після події *keydown* і до події *keyup*.

Потрібно враховувати, що подія *keypress* генерується тільки для тих клавіш, які формують вивід у вигляді символів, зокрема при друці символів. Натискання на інші клавіші, наприклад на *Alt*, не враховується.

Для роботи з подіями клавіатури визначено **об'єкт KeyboardEvent**, який, крім властивостей **об'єкта event**, має набір своїх специфічних для клавіатури **властивостей**:

- *code* повертає рядкове найменування символу під час події натискання клавіші *keypress*. Для літер має вигляд: *"keyD"*, *"keyF"*, такі значення будуть повертатися, незалежно від встановленої мови і регістру. Для цифр верхнього блоку клавіатури повертає значення вигляду *"Digit5"*, для блокового – *"Numpad5"*;
- *charCode* повертає числовий код Unicode натиснутої клавіші під час події натискання клавіші *keypress*;
- *key* повертає символ натиснутої клавіші у вигляді рядка, наприклад *"F"*, *"5"* або *"Enter"*;
- *keyCode* повертає числовий код Unicode натиснутої клавіші під час будь-якої події клавіатури; цей код можна перетворити в символ за допомогою методу *String.fromCharCode*:

  keyStr = String.fromCharCode(event.keyCode);

- *metaKey* повертає *true*, якщо під час генерації події була натиснута метаклавіша клавіатури;
- *altKey* повертає *true*, якщо була натиснута клавіша *Alt* під час генерації події;
- *ctrlKey* повертає *true*, якщо була натиснута клавіша *Ctrl* під час генерації події;
- *shiftKey* повертає *true*, якщо була натиснута клавіша *Shift* під час генерації події.

5.9. Робота з формами

Один із способів взаємодії з користувачами реалізують html-форми. Наприклад, якщо потрібно отримати від користувача деяку інформацію, то можна визначити на веб-сторінці форму, яка буде містити текстові поля для вводу

інформації і кнопку для відправки даних. Після введення даних можна обробити, введену користувачем, інформацію.

Для **створення форми** використовується елемент *form*:

```
<form name = "searchForm">
</form>
```

В JavaScript форма представлена **об'єктом HtmlFormElement**. Після створення форми до неї можна **звернутися кількома способами**.

Перший спосіб полягає в прямому зверненні по імені форми:

```
var sForm = document.searchForm;
```

Другий спосіб полягає у зверненні до колекції форм документа і пошуку в ній потрібної форми:

```
<body>
…
  <form name="searchForm"></form>
  <form name="settingsForm"></form>
…
<script>
 var sForm;
 for (var i = 0; i < document.forms.length; i ++) {
  if (document.forms[i].name === "searchForm")
     sForm = document.forms[i];
 }
 document.write(sForm.name);
</script>
…
</body>
```

Ще один спосіб поєднує обидва підходи:

```
var sForm = document.forms["searchForm"];
```

Також можна застосовувати стандартні способи для пошуку форми, наприклад по *id*, тегу або по селектору:

```
var sForm = document.getElementsByTagname("form")[0];
```

Форма має набір властивостей, з яких найбільш важливими є **властивість *name*** та **властивість *elements***, яка містить колекцію елементів форми:

```
<form name = "searchForm">
  <input type = "text" name = "keyForm"> </input>
```

```
    <input type = "submit" name = "sendForm"> </input>
</form>
<script>
  var sForm = document.forms["searchForm"];
  for (var i = 0; i < sForm.elements.length; i++)
    document.write(sForm.elements[i].name + "<br/>");
</script>
```

Серед методів форми треба відзначити **метод *submit()***, який відправляє дані з форми на сервер, і **метод *reset()***, який очищає поля форми:

```
var sForm = document.forms["searchForm"];
sForm.submit();
sForm.reset();
```

Форма може містити різні **елементи вводу html**: *input*, *textarea*, *button*, *select* і т.д. Усі вони мають ряд загальних властивостей і методів. Також як і форма, елементи форм мають **властивість *name***, за допомогою якої можна отримати значення атрибута *name*:

```
<form name = "searchForm">
  <input type="text" name="keyForm" value="val"> </input>
  <input type="submit" name="sendForm"></input>
</form>
<script>
  var sForm = document.forms["searchForm"];
  // виведемо імена всіх елементів
  for (var i = 0; i < sForm.elements.length; i++)
    document.write (sForm.elements[i].name + "<br/>");
  // отримаємо по імені текстове поле
  var key = sForm.elements["keyForm"];
  document.write(key.name); // keyForm
</script>
```

Іншою важливою властивістю є **властивість *value***, яка дозволяє отримати або змінити значення поля:

```
var sForm = document.forms["searchForm"];
var key = sForm.elements["keyForm"];
document.write(key.value); // val
  // задання значення
key.value = "Привіт світ";
```

За допомогою **властивості *form*** можна отримати батьківський об'єкт – форму:

```
var sForm = document.forms["searchForm"];
var key = sForm.elements["keyForm"];
document.write(key.form.name); // searchForm
```

Ця властивість може бути корисною, наприклад, при відправці даних форми, коли перед безпосередньою відправкою форми необхідно провести валідацію всіх полів форми.

Властивість *type* дозволяє отримати тип поля вводу: це або назва тега елемента (наприклад, *textarea*), або значення атрибуту *type* для елементів *input*. З методів можна виділити **методи *focus()*** (встановлює фокус на елемент) і ***blur()*** (забирає фокус з елемента).

Для відправки введених даних на формі використовуються **кнопки**. Для створення кнопки використовується **елемент *button*** або **елемент *input***:

```
<button name = "send"> Надіслати </button>
<input type = "submit" name = "send" value = "Відправити"/>
```

З точки зору функціональності, в html ці елементи не зовсім рівноцінні, але, в цьому випадку, вони нас цікавлять з точки зору взаємодії з кодом javascript.

При натисканні на будь-який із цих двох варіантів кнопки відбувається відправка даних з форми за адресою, яка вказана в **атрибуті *action* форми**, або за адресою веб-сторінки, якщо атрибут *action* не вказано. Однак **у коді javascript можна перехопити відправку, обробляючи подію *click***:

```
…
<form name="searchForm">
   <input type="text" name="key"></input>
   <input type="submit" name="send" value="Відправити" />
</form>
<script>
 function FormSend(e) {
    / отримуємо значення поля key
  var key = document.searchForm.key;
  var valKey = key.value;
  if (valKey.length > 5) {
     alert("Неприпустиме значення довжини рядка");
```

```
        e.preventDefault();
      }
      else alert("Відправку даних дозволено!");
    }
    var buttonSend = document.searchForm.send;
    buttonSend.addEventListener("click", FormSend);
  </script>
  …
```

При натисканні на кнопку *"Відправити"* виникає подія *click*, і для її обробки, до кнопки прикріплено **обробник *FormSend***. У цьому обробнику перевіряємо введений в текстове поле текст. Якщо його довжина більше 5 символів, то виводимо повідомлення про неприпустиму довжину рядка і перериваємо звичайний хід події за допомогою виклику *e.preventDefault()*. У результаті, дані форми не будуть відправлені. Якщо довжина тексту строго менше шести символів, то виводиться повідомлення *"Відправку даних дозволено!"*, і дані з форми відправляються.

Також можна при необхідності змінити **адресу, на яку відправляються дані, за допомогою властивості *action***:

```
function FormSend(e) {
   // отримуємо значення поля key
   var key = document.searchForm.key;
   var valKey = key.value;
   if (valKey.length > 5) {
      alert("Неприпустима довжина рядка");
      document.searchForm.action = "PostForm";
   }
   else  alert("Відправку даних дозволено!");
}
```

У цьому випадку, якщо довжина тексту більше п’яти символів, то текст відправляється, але за **адресою *PostForm***.

Для **очищення форми** призначені рівноцінні за функціональністю кнопки:

```
<button type = "reset"> Очистити </button>
<input type = "reset" value = "Очистити"/>
```

При натисканні на будь-яку із цих кнопок, відбудеться очищення форми. Функціональність з очищення полів форми

можна реалізувати за допомогою **методу *reset()***:

document.searchForm.reset();

Крім спеціальних кнопок відправки і очищення, на формі можуть використовуватися **звичайні кнопки**:

<input type = "button" name = "send" value = "Відправити"/>

При натисканні на подібну кнопку, відправки даних не відбудеться, хоча також генерується подія *click*. У разі потреби, можна, засобами JavaScript, запрограмувати виконання дій при натисканні на кнопку.

Особливу групу елементів вводу складають **прапорці і перемикачі**. **Прапорець** – це поле, яке може перебувати в одному з двох станів: відміченому або невідміченому. Відмічений стан позначається галочкою. Прапорець створюється за допомогою елемента *<input type = "checkbox"/>*. Важливою рисою прапорця є **властивість *checked***, яка у відміченому стані має значення *true*:

```
<form name = "myForm">
   <input type = "checkbox"
          name = "enabled" checked/> Відмітити
</form>
<div id = "printDiv"> </div>
<script>
  var flag = document.myForm.enabled;
  function onclick(e) {
     var printDiv = document.getElementById("printDiv");
     var enabled = e.target.checked;
     printDiv.textContent = enabled;
  }
  flag.addEventListener("click", onclick);
</script>
```

Натискання на прапорець генерує подію *click*. При обробці цієї події в блок *div* виводиться інформація про відміченість прапорця (вигляду *true/false*).

Перемикачі утворюють групу кнопок, з яких можна вибрати тільки одну. Перемикачі створюються елементом *<input type = "radio"/>*. Вибір або натискання на одну з радіокнопок також генерує подію *click*.

5.10. Media API. Управління відео з JavaScript

Разом з новими елементами *audio* і *video* в HTML5, в JavaScript було додано новий **API** (**A**pplication **P**rogramming **I**nterface) для управління цими елементами. За допомогою коду JavaScript можна отримати елементи *video* і *audio* (як і будь-який інший елемент) і використовувати їх властивості. В JavaScript ці елементи представлені об'єктом **HTMLMediaElement**, який за допомогою властивостей, методів і подій дозволяє управляти відтворенням аудіо та відео.

Найважливі властивості для налаштування цих елементів:

- *playbackRate* задає швидкість відтворення, за замовчуванням дорівнює 1;
- *src* повертає назву відтворюваного ресурсу, якщо він заданий в коді html-елемента;
- *duration* повертає тривалість файлу в секундах;
- *buffered* повертає тривалість тієї частини файлу, яка вже буферизована і готова до відтворення;
- *controls* задає або повертає наявність атрибуту *controls*, якщо він встановлений, повертає *true*, інакше – *false*;
- *loop* задає або повертає наявність атрибуту *loop*, якщо він встановлений, повертає *true*, інакше – *false*;
- *muted* задає або повертає наявність атрибуту *muted*, якщо він встановлений, повертає *true*, інакше – *false*;
- *preload* задає або повертає наявність атрибуту *preload*;
- *volume* задає або повертає рівень звуку від 0.0 до 1.0;
- *currentTime* повертає поточний час відтворення.

Окремо **для елемента *video*** наявні **властивості**:

- *poster* встановлює або повертає атрибут *poster*;
- *height* встановлює або повертає атрибут *height*;
- *width* встановлює або повертає атрибут *width*;
- *videoWidth*, *videoHeight* для елемента *video* повертають ширину і висоту відео відповідно.

Слід також зазначити **два методи**, за допомогою яких можна управляти відтворенням:

- *play()* починає відтворення;
- *pause()* призупиняє відтворення.

Основні події елементів *video* і *audio*:

- *canplaythrough* виникає після завантаження сторінки, якщо браузер визначить, що може відтворювати це відео/аудіо;
- *pause* виникає, якщо відтворення мультимедіа призупиняється, і воно переходить в стан "*paused*";
- *play* виникає, коли починається відтворення файлу;
- *volumechange* виникає при зміні рівня звуку мультимедіа;
- *ended* виникає при завершенні відтворення;
- *timeupdate* виникає при зміні часу відтворення;
- *error* генерується при виникненні помилки;
- *loadeddata* виникає, коли буде завантажений перший кадр відеофайлу;
- *loadedmetadata* виникає після завантаження метаданих мультимедіа (тривалість звучання, розміри відео і т.д.);
- *seeking* виникає, коли користувач починає переміщати курсор по шкалі відтворення для переміщення до нового місця аудіо- або відеофайлу;
- *seeked* виникає, коли користувач завершив переміщення до нового місця по шкалі відтворення.

Використаємо деякі з цих властивостей, методів і подій для управління елементом *video*:

```
<!DOCTYPE html>
<html>
  <head>
     <meta charset="utf-8">
     <title>Відео в HTML5</title>
     <style>
        .hidden{ display:none; }
        #playBtn { border: solid 2px #333;
                   padding: 10px; cursor: pointer; }
     </style>
  </head>
  <body>
   <video width="500" height="400">
        <source src="MyVideo.mp4" type="video/mp4">
        <source src="MyVideo.webm" type="video/webm">
        <source src="MyVideo.ogv" type="video/ogg">
   </video>
```

```
<div id="controls" class="hidden">
    <a id="playBtn">Play</a>
    <span id="timer">00:00</span>
    <input type="range" min="0" max="1" step="0.1"
        value="0" id="volume" />
</div>
<script>
 // отримуємо всі елементи
    var videoElem =
        document.getElementsByTagName('video')[0],
    playBtn = document.getElementById('playBtn'),
    videoControls = document.getElementById('controls'),
    volumeControl = document.getElementById('volume'),
    timePicker = document.getElementById('timer');
 /*якщо браузер може відтворювати відео
   видаляємо клас*/
  videoElem.addEventListener('canplaythrough', function () {
    videoControls.classList.remove('hidden');
    videoElem.volume = volumeControl.value;
  }, false);
  // запускаємо або зупиняємо відтворення
  playBtn.addEventListener('click', function () {
    if (videoElem.paused) {
      videoElem.play();
    } else { videoElem.pause(); }
  }, false);
  videoElem.addEventListener('play', function () {
    playBtn.innerText = "Pause"; }, false);
  videoElem.addEventListener('pause', function () {
    playBtn.innerText = "Play"; }, false);
  volumeControl.addEventListener('input', function () {
    videoElem.volume = volumeControl.value;
  }, false);
  videoElem.addEventListener('ended', function () {
    videoElem.currentTime = 0;
  }, false);
  videoElem.addEventListener('timeupdate', function () {
    timePicker.innerHTML =
```

```
            secondsToTime(videoElem.currentTime);
        }, false);
        // розрахунок відображуваного часу
        function secondsToTime(time){
          var h = Math.floor(time / (60 * 60)),
              dm = time % (60 * 60),
              m = Math.floor(dm / 60),
              ds = dm % 60,
              s = Math.ceil(ds);
            if (s === 60) { s = 0; m = m + 1; }
            if (s < 10) { s = '0' + s; }
            if (m === 60) { m = 0; h = h + 1; }
            if (m < 10) { m = '0' + m; }
            if (h === 0) { Full_time = m + ':' + s;
            } else {
                Full_time = h + ':' + m + ':' + s;
            }
            return Full_time;
        }
    </script>
  </body>
</html>
```

5.11. JSON

JSON (JavaScript Object Notation) – полегшений формат зберігання даних. JSON описує структуру і організацію даних JavaScript. Простота JSON привела до того, що на сьогодні він є найбільш популярним форматом передачі даних у середовищі web, витіснивши інший колись популярний формат XML.

Об’єкти JSON дуже схожі на об’єкти JavaScript, тим більше що JSON є підмножиною JavaScript. У той же час важливо їх розрізняти: JavaScript є мовою програмування, а JSON – форматом даних.

JSON підтримує **три типи даних**: примітивні значення, об’єкти і масиви. **Примітивні значення** – це стандартні рядки, числа, значення *null*, логічні значення *true* і *false*.

Об’єкти являють собою набір примітивних даних, інших об’єктів та масивів. Наприклад, типовий об’єкт JSON:

{ "name": "George", "married": false, "age": 30 }

У JavaScript цьому об’єкту відповідав би наступний:

var user = { name: "George", married: false, age: 30 }

Незважаючи на загальну схожість, є й **відмінності**: в JSON назви властивостей знаходяться в лапках, як звичайні рядки. Крім того, об’єкти JSON не можуть зберігати функції, змінні, як об’єкти JavaScript.

Об’єкти можуть бути складеними:

```
{
   "name": "George",
   "married": false,
   "age": 30,
   "company": { "name": "Google", "address": "USA, LA" }
}
```

Масиви в JSON, схожі на масиви JavaScript і також можуть зберігати найпростіші дані або об’єкти:

["George", true, 30]

Масив об’єктів:

```
[{ "name": "George", "married": false, "age": 21 },
{ "name": "Alex", "married": false, "age": 32 }]
```

Для серіалізації об’єкта JavaScript в JSON застосовується функція *JSON.stringify()*:

```
var person = { name: "George", married: false, age: 21 };
  /* JSON.stringify() перетворює значення JavaScript
    в рядок JSON*/
var serializedPerson = JSON.stringify(person);
document.write(serializedPerson);
  // { "name": "George", "married": false, "age": 21}
```

Для зворотної операції – **десеріалізації або парсингу json-об’єкта в JavaScript** застосовується метод *JSON.parse()*:

```
  // десеріалізація
var person = JSON.parse(serializedPerson);
document.write(person.name); // George
```

5.12. Зберігання даних засобами cookie

Однією з можливостей зберігання даних в JavaScript є **використання cookie**.

Для роботи з cookie в об'єкті document призначена **властивість *cookie***. Для задання cookie достатньо присвоїти властивості *document.cookie* рядок з cookie:

```
<script>
   document.cookie = "login = ivo21;";
</script>
```

У цьому випадку визначається cookie, яка називається *"login"* і має значення *"ivo21"*. Рядок cookie може мати до шести **різних параметрів**: ім'я, значення, термін закінчення дії (expires), шлях (path), домен (domain) і secure. Вище використовувалися тільки два параметри: **ім'я і значення**, тобто у випадку з рядком *"login = ivo21;"* cookie має ім'я *login* і значення "*ivo21"*.

Але cookie такого вигляду має дуже обмежений термін життя: якщо явно не задати термін дії, то cookie буде вилучена із закриттям браузера. Така ситуація ідеальна для тих випадків, коли необхідно видаляти всю інформацію після завершення роботи з веб-додатком і закриттям браузера. Однак така поведінка не завжди підходить. У такому випадку, потрібно встановити параметр *expires*, тобто **термін дії cookie**:

```
document.cookie = "login = ivo21; expires = Wed, 31 Aug 2022
00:00:00 GMT;";
```

Отже, термін дії cookie *login* закінчиться в середу 31 серпня 2022 року в 00:00. Формат параметра *expires* дуже важливий. Його можна згенерувати програмно. Для цього використовують **метод *toUTCString()* об'єкта Date**:

```
var exp = new Date();
exp.setHours(exp.getHours() + 4);
document.cookie=
  ="login=ivo21; expires="+exp.toUTCString()+";";
```

Тепер термін дії cookie становитиме 4 години.

Якщо потрібно **встановити cookie для деякого шляху на сайті**, то можна використовувати параметр *path*. Наприклад, потрібно задати cookie тільки для шляху *www.mysite.com/home*:

document.cookie = "login = ivo21; path =/home;";

У цьому випадку для інших шляхів на сайті, наприклад *www.mysite.com/shop*, ці cookie будуть недоступні.

Якщо на сайті є кілька доменів і потрібно **встановити cookie тільки для певного домену**, тоді можна використовувати параметр *domain*. Наприклад, є піддомен *shop.mysite.com*:

document.cookie = "login = ivo21; expires = Wed, 31 Aug 2022 00:00:00 GMT; path = /; domain = shop.mysite.com;";

Параметр *path=/* вказує, що cookie будуть доступні для всіх директорій і шляхів піддомену *shop.mysite.com*.

Параметр *secure* задає **використання SSL** (SecureSockets Layer) і підходить для сайтів, які використовують протокол https. Якщо значення цього параметру дорівнює *true*, то cookie будуть використовуватися тільки при встановленні захищеного з'єднання SSL. За замовчуванням, цей параметр дорівнює *false*.

document.cookie = "login = ivo21; expires = Wed, 31 Aug 2022 00:00:00 GMT; path = /; domain = shop.mysite.com; secure = true;";

Для найпростішого читання cookie з браузера достатньо звернутися до властивості *document.cookie*:

```
var exp = new Date();
exp.setHours(exp.getHours() + 4);
document.cookie=
    "city=Paris; expires ="+exp.toUTCString()+";";
document.cookie=
    "country=France; expires ="+exp.toUTCString()+";";
document.cookie = "login = ivo21;";
document.write(document.cookie);
```

Тут було встановлено три cookie, і браузер виведе на сторінку інформацію про них.

Отримані cookie не включають параметри *expires*, *path*, *domain* і *secure*. Крім того, **самі cookie розділяються крапкою з комою**, тому потрібно ще провести деякі перетворення, щоб отримати їх ім'я і значення:

```
var items = document.cookie.split(";");
for (var i = 0; i < items.length; i++) {
   var item = items[i].split("="),
```

```
    name = item[0],
    value = item[1];
    document.write("Ім'я:" + name + "<br/>");
    document.write("Значення:" + value + "<br/> <br/>");
}
```

Хоча cookie дозволяють зберігати інформацію, вони мають **ряд обмежень**. Наприклад, браузер має обмеження на розмір cookie – кожна cookie не може перевищувати 4кб. Cookie мають термін дії, після якого видаляються. Cookie є невід'ємною рисою протоколу HTTP і при кожному запиті до сервера передаються разом із запитом на сервер. Однак для роботи з cookie, на стороні клієнта, в коді JavaScript не має значення передача cookie на сервер. Крім того, для отримання збережених cookie потрібно написати певний код.

Тому в HTML5 була впроваджена нова концепція зберігання даних – **web storage**. Web storage складається з двох компонентів: session storage і local storage.

1. **Session storage** реалізує тимчасове сховище інформації, яка видаляється після закриття браузера.

2. **Local storage** – сховище даних на постійній основі. Дані з local storage автоматично не видаляються і не мають терміну дії. Ці дані не передаються на сервер в запиті HTTP. Крім того, обсяг local storage становить в Chrome і Firefox 5 Мб для домену, а в IE – 10 Мб.

Всі дані у web storage являють собою **набір пар ключ-значення**, тобто кожний об'єкт має унікальне ім'я-ключ і певне значення.

Для роботи з local storage в JavaScript використовується **об'єкт localStorage**, а для роботи з session storage – **об'єкт sessionStorage**.

Для збереження даних вони повинні бути передані в метод *setItem()* об'єкта localStorage:

```
localStorage.setItem ( "login", "ivo21@gmail.com");
```

У цей метод передаються два значення: ключ і значення об'єкта, що зберігається. Якщо в localStorage вже є об'єкт з ключем *"login"*, то його значення замінюється новим. **Для отримання збережених даних** використовується метод *getItem()*:

var login = localStorage.getItem("login"); //ivo21@gmail.com

У цей метод передається ключ об'єкта.

Щоб видалити об'єкт, застосовується метод *removeItem()*, який приймає параметр – ключ об'єкта, що видаляється:

localStorage.removeItem("login");

Для повного видалення всіх об'єктів з localStorage можна використовувати метод *clear()*:

localStorage.clear();

Зі збереженням простих об'єктів усе зрозуміло. Однак при цьому потрібно враховувати, що дані в localStorage зберігаються у вигляді рядка:

```
localStorage.setItem("age", 23);
var age = localStorage.getItem("age");
age = parseInt(age) +10;
document.write(age); // 33
```

Якщо, в цьому випадку, не перетворити рядкове значення до числа за допомогою *parseInt()*, то *age* буде використовуватись як рядок.

Труднощі можуть виникнути **зі збереженням складених об'єктів**:

```
var person = { name: "George", age: 23, married: false };
localStorage.setItem("person", person);
var savedPerson = localStorage.getItem("person");
document.write(savedPerson); // [object Object]
document.write(savedPerson.name); // undefined
  // undefined, оскільки savedPerson – рядок, а не об'єкт
```

У цьому випадку потрібно використовувати серіалізацію в формат JSON:

```
var person = { name: "George", age: 23, married: false };
localStorage.setItem("person", JSON.stringify(person));
var savedPerson =
    JSON.parse(localStorage.getItem("person"));
document.write(savedPerson.name+" "+savedPerson.age+" "+
    savedPerson.married); // George 23 false
```

На завершення зазначимо, що в деяких браузерах, наприклад в Google Chrome, за допомогою спеціальних інструментів можна побачити збережені об'єкти в local storage.

6. БІБЛІОТЕКА JQUERY

6.1. Загальні відомості та підключення jQuery

Сучасне веб-програмування та створення веб-сайтів неможливо уявити без використання мови JavaScript. Однак в наш час все частіше використовується не «чистий» код javascript, а javascript-фреймворки і бібліотеки. **Однією з таких бібліотек**, причому, можливо, найпопулярнішою на сьогодні, є **jQuery**. За деякими оцінками, не менше половини великих сайтів в інтернеті використовують цю бібліотеку.

Хоча ми називаємо jQuery бібліотекою, але насправді поняття «jQuery» об'єднує цілу екосистему бібліотек, побудовану навколо базової бібліотеки: це і бібліотека **jquery.ui**, призначена для створення візуальних інтерфейсів, і **jquery.mobile**, використовувана при розробці мобільних сайтів та ін.

Переваги використання jQuery:

1. **Спрощення роботи з кодом**: jQuery пропонує простий елегантний синтаксис для маніпулювання елементами на веб-сторінці.

2. **Можливість розширення**: весь код jQuery відкритий для перегляду і зміни, якщо щось у бібліотеці не влаштовує, її можна модифікувати, також є можливість створювати плагіни jQuery.

3. **Кросбраузерність**: jQuery має підтримку більшості відомих браузерів, у тому числі таких, як IE 7,8. Хоча у зв'язку з тим, що браузери IE 6-8 поступово стають надбанням історії, і щоб зменшити розмір бібліотеки, в останній версії була припинена підтримка IE 6-8.

Щоб почати працювати із цією бібліотекою, потрібно, насамперед, її завантажити з офіційного сайту розробника ***https://jquery.com/download/***. На сторінці завантажень можна знайти різні версії jQuery. Хоча версії дещо відрізняються, але ці відмінності, як правило, не настільки суттєві, і базовий стержень та загальні принципи у більшості версій фактично одні й ті самі.

Бібліотека представлена в двох варіантах: **Compressed або Monified** (мінімізована) і **Uncompressed** (звичайна).

Мінімізовані версії надають ту ж функціональність, що й звичайні, але відрізняються тим, що не містять необов'язкових символів типу пробілів, коментарів і т.д. Тому в своїй назві мають **суфікс *min***, наприклад *jquery-1.10.1.min.js*. Оскільки ці версії продуктивніші за рахунок меншого обсягу, їх рекомендується використовувати у реальному виробництві. В той же час якщо потрібно зрозуміти логіку коду jQuery, то можна звертатися до звичайної версії бібліотеки.

Бібліотека jQuery підключається, аналогічно, як і інші js-файли:

```
<script src="jquery-1.10.1.min.js"></script>
```

Створимо найпростішу веб-сторінку з використанням jquery:

```
<!DOCTYPE html>
<html>
<head>
    <title>jQuery</title>
    <script src="jquery-1.10.1.min.js"></script>
</head>
<body>
    <h2> jQuery </h2>
    <button id="btn1">jQuery</button>
    <button id="btn2" onclick="alert('JavaScript');">
      JavaScript
    </button>
    <script type="text/javascript">
        $(function(){
            $('#btn1').click(function(){
                $(this).css('background-color', 'red');
                alert('jQuery');
            });
        });
    </script>
</body>
</html>
```

Ця веб-сторінка, з одного боку, застосовує код jquery, з іншого – демонструє відміності від використання стандартного коду JavaScript. На сторінці визначені **дві кнопки**. Для другої

кнопки визначено обробник *onclick* у самій розмітці кнопки: *onclick = "alert('JavaScript');"*. Інша кнопка робить практично те ж саме, але з використанням jQuery. Для неї визначено *id* (*id = "btn1"*), використовуючи який, керуємо кнопкою у функції jQuery.

Вираз ***$(function(){…});*** і є **коротке визначення функції JQuery**. Цю функцію прийнято розміщувати в кінці документа, як у даному випадку, перед закриваючим тегом *</body>*. Функція включає весь код JavaScript, який буде виконуватися під час завантаження сторінки.

Функція jQuery має наступний синтаксис:

```
jQuery(document).ready(function(){
  // код функції
});
```

Формально, опис функції jQuery*: jQuery(об'єкт)*. У даному випадку як об'єкт використовується об'єкт *document*, що представляє практично всю структуру DOM веб-сторінки. І до нього застосовується обробник *ready*, який сигналізує про те, що DOM-модель веб-сторінки завантажена. Як параметр обробника використовується безіменна функція зворотного виклику, яка і спрацьовує при завантаженні веб-сторінки.

Тобто, фактично, говоримо веб-браузеру, що після завантаження всієї об'єктної моделі веб-сторінки, представленої об'єктом document, він повинен виконати весь код, який ми вкладаємо у функцію jQuery.

Є ще один **спосіб оголошення**, який також рівнозначний попередньому:

```
$(document).ready(function(){
  // код функції
});
```

Знак $ є псевдонімом jQuery.

Але можна використовувати і **скорочені записи функції jQuery**:

```
$(function(){
  // код функції
});
```

Або так:

```
jQuery(function(){
```

// код функції
});
Наведені чотири форми рівнозначні.

6.2. Селектори та фільтри

Однією з важливих функціональностей jQuery є вибірка елементів. Функція jQuery може повертати різні елементи DOM. Щоб щось з ними робити, потрібно їх спочатку отримати. Бібліотека надає зручний **спосіб вибору елементів, що базується на селекторах**. Достатньо передати у функцію jQuery селектор, і ми можемо отримати потрібний елемент, який відповідає цьому селектору. Наприклад, щоб отримати всі елементи *img*, використовують вираз: *$('img')*, де *img* виступає як селектор.

Розглянемо **основні селектори JQuery**:

- *$('*')* – вибір усіх елементів сторінки;
- *$('div')* – всі елементи *div*;
- *$('#sidebar')* – всі елементи зі значенням ідентифікатора *“sidebar”* (*id="sidebar"*);
- *$('.post')* – всі елементи з класом *'post'*;
- *$('div.post')* – всі елементи *div* із класом *'post'*;
- *$('div span')* – всі елементи *span* всередині *div* на всіх рівнях вкладеності;
- *$('div>span')* – всі елементи *span* всередині *div*, які є прямими нащадками *div* (перший рівень вкладеності);
- *$('div, span')* – всі *div* і *span* одночасно;
- *$('div+span')* – перший *span* після *div*, у випадку, якщо цей елемент *span* знаходиться безпосередньо після *div*;
- *$('#banner').prev()* – елемент перед *#banner*;
- *$('#banner').next()* – елемент після *#banner*;
- *$('div').find('span')* – всі елементи *span* всередині *div* (на всіх рівнів вкладеності);
- *$('p>*')* – всі нащадки елемента *p* (перший рівень вкладеності);
- *$('p').children()* – всі нащадки елемента *p* (перший рівень вкладеності);

- *$('p').parent()* – прямий предок елемента *p* (строго на один рівень вище);
- *$('p').parents()* – всі предки елемента *p* (всіх рівнів вище);
- *$('p').parents('div')* – всі предки *p*, які є елементами *div* (всіх рівнів вище).

Крім цього, можна використовувати **фільтри**, які дозволяють конкретизувати вибірку елементів і зробити селектори більш гнучкими:

- *$('div:first')* – перший, у вибірці, елемент *div*;
- *$('div:last')* – останній, у вибірці, елемент *div*;
- *$('div:not(.post)')* – елементи *div*, у яких атрибут *class* не дорівнює *'post'*;
- *$('div:even')* – елементи *div* з парними номерами;
- *$('div:odd')* – елементи *div* з непарними номерами;
- *$('div:eq(3)')* – елемент *div*, 3-й по рахунку (нумерація з нуля);
- *$('div:gt(3)')* – елементи *div* з порядковим номером >3;
- *$('div:lt(3)')*– елементи *div* з порядковим номером < 3;
- *$(':header')* – всі заголовки *h1*, *h2*, *h3* і т.д.;
- *$('div:contains("Copyright")')* – всі елементи *div*, які містять текст *"Copyright"*;
- *$('div:empty')* – порожні *div* (які не мають дочірніх елементів);
- *$('div:has(p)')* – всі *div*, які містять елемент *p*;
- *$('div').filter('.post')* – всі *div* з класом *'post'*;
- *$('div:hidden')* – приховані елементи *div*;
- *$('div:visible')* – видимі елементи *div*;
- *$('div[class]')* – всі *div* з атрибутом *class*;
- *$("div[title = 'title']")* – всі *div* з *title='title'*;
- *$("div[title != 'title']")* – всі *div* з *title!= 'title'*;
- *$("div[title ^= 'ti']")* – всі *div* з атрибутом *title*, які починаються з *'ti'*;
- *$("div[title $= 'le']")* – всі *div* з атрибутом *title*, які закінчуються на *'le'*;
- *$("div[title*='itl']")* – всі *div* з атрибутом *title*, що містять *'itl'*;

- *$(':text')* – всі *input* з типом *text*;
- *$(':radio')* – всі *input* з типом *radio*;
- *$('input:enabled')* – всі доступні елементи *input*;
- *$('input:checked')* – всі відмічені чекбокси;
- *$("div[title = 'title']:visible:has(p)")* – *div* з атрибутом *title='title'*, видимий та такий, що містить тег *p*;
- *$('form select option:selected')* – вибрані елементи *select*;
- *$('form:radio[name=some]:checked').val()* – отримання вибраного значення радіобатона з ім'ям *some*;
- *$('form:checkbox:checked')* – вибір всіх відмічених чекбоксів.

6.3. Методи для роботи з DOM

Основними методами бібліотеки jQuery для роботи з DOM є:

- *css('property', 'value')* задає значення властивості CSS;
- *addClass('class')* додає клас;
- *hasClass('class')* перевіряє існування класу;
- *toggleClass('class')*, якщо клас існує, то видаляє, інакше – додає;
- *attr('attribute')* додає атрибут;
- *attr('attribute', 'value')* додає значення атрибуту;
- *removeAttr('attribute')* видаляє атрибут;
- *html('text')* задає чи перезаписує текст всередині html-тегу;
- *html()* отримує вмістиме html-тегу;
- *val()* отримує вмістиме елемента html-форми;
- *after(' text ')* додає текст або html-код (у цьому випадку, * text *), після вказаного елемента;
- *before(' text ')* те саме, що і *after*, але перед вказаним елементом;
- *append('<code>This is new code</code>')* додає вказаний, в дужках, елемент в кінець вмістимого визначеного елемента;
- *prepend('<code>This is new code</code>')* те саме, що і *append*, але елемент буде доданий в початок вмістимого;
- *empty()* очищає вмістиме елемента;

• *remove()* видаляє елемент;

• *wrap('<a href="#">')* обгортає (розміщує) вибраний елемент в елемент-обгортку, який передається як параметр (в даному випадку, *<a href="#">).*

Усі ці методи застосовуються до елементів веб-сторінки, які вибрані з допомогою селектора або фільтра, **наприклад**:

• *$('p').css('border', '3px solid blue');* для всіх елементів *p* (абзаців) задається значення границі *'3px solid blue'*;

• *if($('div').hasClass('paragraph')){ alert('Element has class'); }* перевіряються всі елементи *div* на наявність класу, після чого виводиться відповідне повідомлення;

• *$('.paragraph').attr('class', 'red_border');* для всіх елементів із класом *'paragraph'* додається новий клас *'red_border'*;

• *$('.paragraph').html('text');* для всіх елементів із класом *'paragraph'* перезаписується вмістиме html-елемента на *'text'*.

У межах цього посібника звернемо увагу на **функцію *each()***. Вона забезпечує виконання функції для кожного з обраних елементів окремо. Це дає можливість обробляти вибрані елементи окремо один від одного. Метод має такий синтаксис:

```
$('елемент / фільтр').each (function(index, element))
  {тіло функції}
```

У функцію передаються два параметри: номер елемента в наборі (нумерація починається з нуля) і сам елемент у вигляді об'єкта DOM. Приклад:

```
var heights=[]; // масив значень висоти кожного елемента
$ ('div').each (function(index, element) {
    heights.push ($(element).height());
    console.log(heights[index]);
});
```

У результаті в змінну *heights* будуть розміщені значення висот усіх *div*-елементів.

Крім цього, можна використовувати функцію, не передаючи індекс і значення. При цьому параметр *this* допомагає визначити, який елемент DOM в даний момент обробляється циклом.

Спрощений варіант виглядає так:

```
$('елемент / фільтр').each(function() {
    console.log ($(this).text());
});
```

6.4. Події

Події визначають, при яких умовах виконуються ті чи інші обробники подій. Події прийнято розподіляти на наступні типи:

Події браузера:

- *resize* – зміна розмірів елемента;
- *scroll* – скроллінг елемента;
- *load* – завантаження елемента (*img*);
- *unload* – вигрузка елемента або перехід на другу сторінку (window).

Події форми:

- *change* – зміна значення елемента (значення елемента, при втраті фокусу, відрізняється від початкового, при отриманні фокусу);
- *select* – виділення тексту (актуально тільки для *input[type=text]* і *textarea*);
- *submit* – надсилання даних з форми;
- *blur* – фокус з елемента, актуально для *input[type=text]*, спрацьовує при кліку на іншому елементі сторінки;
- *focus* – фокус на елементі, актуально для *input[type=text]*, але в сучасних браузерах працює і з іншими елементами;
- *focusin* – фокус на елементі, спрацьовує на предку елемента, для якого відбулась подія *focus*;
- *focusout* – фокус з елемента, спрацьовує на предку елемента, для якого відбулась подія *blur*.

Події клавіатури:

- *keydown* – натискання клавіші на клавіатурі;
- *keypress* – натискання клавіші на клавіатурі (порядок подій: *keydown*, *keypress*, *keyup*);
- *keyup* – відпустити клавішу на клавіатурі.

Події миші:

- *click* – клік по елементу (порядок подій: *mousedown*, *mouseup*, *click*);

- *dblclick* — подвійний клік по елементу;
- *mouseout* – вивід курсора з елемента;
- *mousedown* – натискання клавіші миші на елементі;
- *mouseup* – відпустити клавішу миші;
- *mousemove* – рух курсора в межах елемента;
- *mouseenter* – рух курсора на елемент (не спрацьовує при переході фокусу на дочірні елементи; спрацює, коли вказівник миші заходить на елемент);
- *mouseleave* – вивід курсора з елемента (не спрацьовує при переході фокусу на дочірні елементи);
- *mouseover* – наведення курсора на елемент.

6.5. Анімації

Для створення анімації існують такі **функції**:

- *hide()* дозволяє сховати елемент;
- *show()* дозволяє відобразити елемент;
- *toggle()*, якщо елемент прихований, дозволяє відобразити, та приховати, якщо відображений;
- *slideUp()* та *slideDown()* – приховання та відображення елемента з використанням ефекту у вигляді слайду;
- *slideToggle()* – перемикач слайдів об'єднує дві вищеперераховані функції;
- *fadeIn(), fadeOut(), fadeToggle()* – відображення/приховання/перемикання елементів з ефектом затухання (плавною зміною прозорості).

Розглянемо приклад:

```
$(document).ready(function(){
    $('button').click(function(){
    $('p').hide();
    });
});
```

У наведеному прикладі при кліку на елемент *'button'*, причому на будь-який, якщо їх декілька, будуть приховуватися всі абзаци (*p*).

Ще однією важливою функцією є **функція *animate***:

```
animate(properties, [duration], [easing], [callback]);
```

Ця функція через заданий проміжок часу змінює значення вказаних властивостей CSS. **Параметр *duration*** визначає рядкове або числове значення тривалості анімації. Значення, за замовчуванням, для *duration* дорівнює 400 (в мілісекундах). Рядкові ключові слова *'fast'* і *'slow'* відповідають значенням, відповідно, 200 та 600 мілісекунд (великі значення визначають повільну анімацію, а менші – швидку).

Параметр *easing* визначає криву швидкості для анімації (використовується математична функція – кубічна крива Без'є). Без використання зовнішніх плагінів має тільки два значення: *linear* (однакова швидкість від початку до кінця) і *swing* (ефект анімації має повільний старт і повільне завершення, але швидкість збільшується всередині анімації). Значення, за замовчуванням, *swing*.

Параметр *callback* – це функція, яка буде виконана після завершення анімації, вона викликається один раз для кожного відповідного елемента. Всередині функції змінна *this* посилається на DOM-елемент, до якого застосовується анімація.

Можно також запускати ланцюжки анімації:

```
$(document).ready(function(){
    $('button').click(function(){
                $('.red_border').animate({'fontSize':'30px',
                                'borderLeftWidth':'10px',
                                        'padding':'20px'},
            5000, 'linear').animate({'opacity':0.5},
            1000, 'linear', function(){
               console.log("Animation complite")
            });
    });
 });
```

У наведеному прикладі для всіх елементів із класом *'red_border'* з періодом 5 секунд зміняться властивості, до визначених в коді *('fontSize':'30px', 'borderLeftWidth':'10px', 'padding':'20px')* згідно з кривою Без'є зі значенням *'linear'*. Після цього буде застосована наступна анімація, яка буде тривати тільки секунду і по завершенні якої в консоль буде виведено повідомлення про завершення анімації.

7. МОВА ПРОГРАМУВАННЯ PHP

7.1. Основи мови PHP

7.1.1. Вступ

На сьогодні PHP є однією з найбільш поширених мов веб-програмування. Велика кількість сайтів і веб-сервісів в мережі Інтернет створено засобами PHP. За деякими оцінками, PHP використовується більш ніж для 80% сайтів, серед яких такі сервіси, як facebook.com, vk.com, baidu.com та інші. Така популярність невипадкова. Простота мови дозволяє швидко та легко створювати сайти різної складності.

Переваги мови програмування PHP:

1. Для всіх найбільш поширених операційних систем (Windows, MacOS, Linux) є свої версії пакетів розробки на PHP, а це означає, що можна створювати веб-сайти в будь-якій із цих операційних систем.

2. Мова PHP може працювати у зв'язці з різними веб-серверами: Apache, Nginx, IIS.

3. Простота і легкість освоєння мови. Як правило, вже маючи невеликий досвід у програмуванні на PHP, можна створювати прості веб-сайти.

4. Мова PHP схожа на мову C, тому, знаючи C або одну з C-подібних мов, легше освоїти PHP.

5. PHP підтримує роботу з великою кількістю систем баз даних (MySQL, MSSQL, Oracle, PostgreSQL, MongoDB та інші).

6. Поширеність хостингових послуг та їх дешевизна. Оскільки, як правило, хостингові компанії розміщують веб-сайти, створені з використанням PHP, на веб-серверах Apache або Nginx, які працюють на одній з операційних систем сімейства Linux. І веб-сервери, і операційні системи на базі Linux безкоштовні, що знижує загальну вартість використання хостингу.

7. Постійний розвиток. PHP продовжує розвиватися, випускаються нові версії, які мають нові функції, адаптуючи мову програмування до нових викликів сучасності. І, як правило, перейти на нову версію не складає труднощів.

7.1.2. Перший сайт

Для створення сайтів потрібно встановити веб-сервер (Apache,…), мову програмування PHP та СКБД при потребі. Є різні способи встановлення всього необхідного програмного забезпечення. Можна встановлювати **всі компоненти окремо**, а можна використовувати вже **готові збірки** (OpenServer, Denwer, XAMPP, EasyPHP, …). У таких збірках компоненти вже мають початкові налаштування та готові до створення сайтів. Однак рано чи пізно розробникам все одно доводиться вдаватися до встановлення та конфігурації окремих компонентів, підключення інших модулів. Але, оскільки процес окремого підключення та налаштування інтерпретатора PHP, веб-сервера та СКБД детально описаний в багатьох інтернет-джерелах, то, в даному випадку, ми це опускаємо, а в якості ПЗ **візьмемо OpenServer та ОС Windows 10**. Після завантаження програмного забезпечення його потрібно запустити, натиснувши на exe-файл в корені завантаженого архіву. Успішний запуск програмного забезпечення буде відображатися значком в системному трею, натиснення лівої кнопки миші на якому розгортає меню для запуску OpenServer'а, а в подальшому для перегляду своїх сайтів, які будуть доступні в меню «*Мої проекти*». Крім цього, через меню можна проводити різні налаштування програмного забезпечення, зокрема, вибір версій PHP, Apache, СКБД, створення, видалення даних з баз даних через ПЗ PhpMyAdmin, PHPAdminer тощо.

Для того, щоб сайт відображався в моїх проектах, потрібно через меню OpenServer'а, в його директорії «*domains*» створити папку з назвою, яка буде виступати в ролі URL-адреси. Остання має бути з неіснуючим доменом, наприклад: *passenger.traffic*, в якій потрібно створити файл ***index.php*** або ***index.html***. За замовчуванням, обидва ці файли можуть відповідати за виведення головної сторінки сайту. **Php-файл** має вищий пріоритет, тому за наявності двох файлів саме php-файл буде головним файлом сайту. У випадку наявності **тільки html-файлу** він стає основним файлом сайту, а його вмістиме з врахуванням мови HTML виводиметься у вигляді головної сторінки веб-сайту.

Опишемо більш детально **створення невеликого сайту**, призначення якого – дати початкове розуміння роботи з PHP. Для створення програм на PHP можна використати текстовий редактор. Найбільш популярний на сьогоднішній день – Notepad++.

Перейдемо до раніше створеного каталогу сайту в директорії *domains*, який і буде зберігати всі документи сайту. Створимо текстовий файл з назвою ***index.html***. Відкриємо його в текстовому редакторі та наповнимо наступним кодом:

```
<!DOCTYPE html>
<html>
<head>
    <title> Перший сайт на PHP </title>
    <meta charset = "utf-8">
</head>
<body>
    <h2> Введи свої дані: </h2>
    <form action = "display.php" method = "POST">
        <p>Ім'я:<input type = "text" name = "firstname"/>
        </p>
        <p>Прізвище:<input type="text" name="lastname"/>
        </p>
        <input type = "submit" value = "Відправити">
    </form>
</body>
</html>
```

Код html містить форму з двома текстовими полями. При натисканні на кнопку *"Відправити"* дані з форми надсилаються скрипту *display.php*, оскільки він вказаний **в атрибуті *action***.

Тепер **створимо скрипт, який буде обробляти дані з форми**. Додамо в папку сайту новий текстовий файл. Перейменуємо його у ***display.php***. За замовчуванням, файли програм на php мають розширення ***.php***. Додамо у файл *display.php* такий код:

```
<!DOCTYPE html>
<html>
<head>
    <title> Перший сайт на PHP </title>
```

```
    <meta charset = "utf-8">
</head>
<body>
<?php
    $name = $_POST["firstname"];
    $surname = $_POST["lastname"];
    echo "Ваше ім'я: <b>". $name. " ". $surname. "</ b>";
?>
</body>
</html>
```

Тут у html-розмітці міститься код PHP. **Для додавання php-коду на сторінку** використовується тег ***<?php .....?>***, у якому розміщуються оператори мови PHP. У php-коді ми отримуємо дані з форми і виводимо їх на сторінку.

Кожний вираз PHP має завершуватися крапкою з комою. У наведеному прикладі є три вирази. Два з них отримують передані дані з форми, наприклад

$name = $_POST["firstname"];.

Змінна *$name* – це змінна, яка буде зберігати значення. Всі змінні в PHP починаються зі знака «***$***». Оскільки форма на сторінці *index.html* використовує для відправки даних метод ***POST***, то за допомогою виразу ***$_POST["firstname"]*** можна отримати значення, яке було введено в текстове поле з атрибутом ***name=”firstname”***. Це значення зберігається в змінній *$name*.

За допомогою **оператора *echo*** можна вивести на сторінку будь-яке значення або текст, які зазначені після оператора в лапках. У цьому випадку (*echo "Ваше ім'я: ". $name." ". $surname. ""*) за допомогою операції конкатенації (“**.**”) текст в лапках з’єднується зі значеннями змінних *$name*, *$surname* та виводиться на сторінку.

Запуск створеного сайту можна виконати через меню «Мої проекти» OpenServer’а. На головній сторінці сайту розміщується форма для вводу імені та прізвища (рис. 7.1), а після натискання на кнопку «*Відправити*» дані будуть оброблені скриптом *display.php*, який, в свою чергу, виведе прізвище та ім’я, введені у форму на головній сторінці сайту (рис. 7.2).

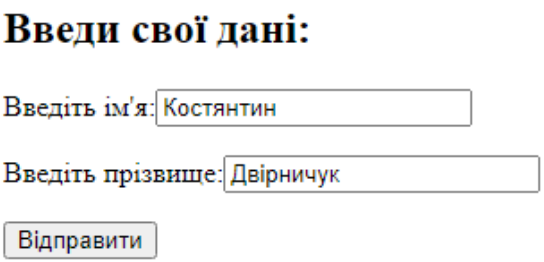


Рис. 7.1. Форма для введення даних

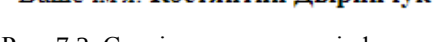


Рис. 7.2. Сторінка виведення інформації

7.1.3. Основи синтаксису

При створенні першої програми на PHP коротко було сказано деякі основні принципи створення скриптів на мові PHP. Тепер розглянемо їх детальніше.

Програма або скрипт на PHP, як правило, знаходиться **в файлі з розширенням .php**, хоча розробники можуть також **вставляти код php і в файли з розширеннями .html/.htm**.

Коли користувач звертається до файлу з php-скриптом в адресному рядку браузера, то веб-сервер передає його інтерпретатору PHP. Інтерпретатор обробляє код і на його основі генерує html-розмітку. Згенерований html-код відправляється користувачу.

PHP-файл може містити як розмітку html, так і код на мові php. Для переходу від розмітки html до коду php використовуються **тег <?php … ?>**, в якому розміщується php-код. Цег тег позначає для інтерпретатора, що його вміст треба інтерпретувати як php-код, а не розмітку html.

Розглянемо найпростіший скрипт на php:

```
…
<?php
    echo "<p> Привіт світ! </p>";
    echo "2 + 2 =". (2 + 2);
?>
…
```

Після обробки файлу інтерпретатор сформує таку розмітку:

…

* Привіт світ! *

2 + 2 = 4

…

Тут використані два оператори *echo " Привіт світ! "* та *echo "2 + 2 =". (2 + 2),* які виводять певне значення на сторінку. Кожний оператор в PHP завершується **крапкою з комою**.

При створенні веб-сайту можна використовувати **коментарі**. Це дає можливість, для зручності, прокоментувати будь-яку дію:

<?php

echo " Привіт світ! "; // вивід повідомлення

?>

Знак «//» визначає **однорядковий коментар**, все що після нього в тому ж рядку, буде вважатися коментарем і не буде виконуватися інтерпретатором. При обробці інтерпретатор буде пропускати коментарі. Якщо потрібно закоментувати кілька рядків, то використовують **багаторядковий коментар** */* текст коментаря */*. Всі рядки всередині багаторядкового коментаря також не обробляються інтерпретатором.

7.1.4. Змінні в PHP та типи даних

Змінні в PHP, як і в більшості мов програмування, призначені для зберігання окремих значень та їх використання у виразах. Для позначення змінної використовується **знак долара $**, наприклад: *$a = 10;* що визначає змінну, яка зберігає число 10. Присвоєння значення відбувається за допомогою знака рівності =. Змінній можна присвоїти значення іншої змінної.

PHP є **чутливою до регістру мовою**, це означає, що змінні *$counter* і *$Counter* являють собою дві різні змінні.

При найменуванні змінних діють такі правила:

- імена змінних повинні починатися з алфавітного символу або зі знаку підкреслення;
- імена змінних можуть містити тільки символи: a-z, A-Z, 0-9, і знак підкреслення;

• імена змінних не можуть містити пробіли.

Якщо змінна оголошена, але їй не присвоєно жодного значення (інакше кажучи, вона не ініціалізована), то буде проблемно її використовувати. Наприклад:

```
$a;
echo $a;
```

при спробі вивести значення неініціалізованої змінної, отримаємо повідомлення, що змінна не визначена:

Notice: Undefined variable: a in C: \ localhost \ echo.php on line 3.

Оператор *isset()* перевіряє, чи ініціалізована змінна, значенням, відмінним від *NULL*. Якщо змінна ініціалізована, то *isset()* повертає *true*, в протилежному випадку – *false*. За допомогою **оператора *unset()*** можна знищити змінну:

```
$a = 20;
if (isset($a)) echo $a; //20
unset($a);
echo $a; // помилка, змінна не визначена
```

PHP є мовою з динамічною типізацією. Це означає, що тип даних змінної визначається під час виконання і, на відміну від ряду інших мов програмування, в PHP не потрібно вказувати перед змінною тип даних.

PHP підтримує **вісім простих типів даних**:

- *boolean* (логічний тип): *true* або *false*;
- *integer* (цілі числа);
- *double* (дробові числа);
- *string* (рядки);
- *array* (масиви);
- *object* (об'єкти);
- *resource* (ресурси);
- *NULL*.

Цілочисловий тип (***integer***) представляє ціле число зі знаком розміром 32 біти (від -2 147 483 648 до 2 147 483 647). Крім десяткових цілих чисел, PHP має можливість використовувати також двійкові, вісімкові та шістнадцяткові числа.

Розмір числа з плаваючою крапкою залежить від платформи. Максимально можливе значення, як правило, становить ~ 1.8e308 з точністю близько 14 десяткових цифр.

Змінні логічного типу (***boolean***) мають два можливі значення: *true* і *false* або, інакше кажучи, істинно і хибно.

Значення *NULL* означає, що значення змінної не визначено. Буває корисним у тих випадках, коли потрібно вказати, що змінна не має значення. Наприклад, якщо створити змінну без ініціалізації і її використати, то інтерпретатор видасть повідомлення, що змінна не визначена. Використання значення *NULL* дозволяє уникнути такої ситуації. Крім того, можна перевірити наявність значення і, в залежності від результату перевірки, виконувати ті чи інші дії:

```
$a = NULL;
if ($a)  echo "Змінна a визначена";
else  echo "Змінна a не визначена";
```

Результуючий вивід: *«змінна a не визначена”*

Для роботи з текстом використовуються **рядки**. Рядки бувають **двох типів**: в подвійних лапках і одинарних. Від типу лапок залежить обробка рядків інтерпретатором. Наприклад, замість змінних в подвійних лапках підставляються їх значення, а змінні в одинарних лапках залишаються незмінними:

```
$number1 = 10;
$number2 = 5;
$res = "$number1 + $number2 <br>";
echo $res;
$res = '$number1 + $number2';
echo $res;
```

У цьому випадку, отримаємо наступний вивід:

```
10 + 5
$number1 + $number2
```

Крім звичайних символів, рядок може містити **спеціальні символи**, які можуть бути неправильно інтерпретовані. Наприклад, потрібно додати в рядок лапки:

```
$text = "Модель "Apple II"";
```

Цей запис некоректний, і компілятор видасть помилку *syntax error*… . Щоб це виправити, поєднують **різні типи лапок** (*'Модель "Apple II"'* або *"Модель 'Apple III'"*) або

використовують слеш, щоб ввести лапки в рядок: *$text = "Модель \" Apple II \"";*

Ресурс (resource) – це спеціальна змінна, що містить посилання на зовнішній ресурс. Зовнішнім ресурсом може бути, наприклад, файл або підключення до бази даних. Ресурси створюються і використовуються спеціальними функціями. Далі ми докладніше розглянемо роботу з файлами та підключення до бази даних.

Описом об'єкта є **клас**, а **об'єкт** являє собою екземпляр цього класу. Можна провести таку аналогію: у всіх є деяке уявлення про людину – наявність двох рук, двох ніг, голови, травної, нервової системи, головного мозку і т.д. Тобто є деякий шаблон – цей шаблон можна назвати класом. А реально існуюча людина (фактично, екземпляр даного класу) є об'єктом цього класу.

Масиви будуть розглянуті у відповідному розділі посібника.

Константи, як і змінні, зберігають певне значення, але, на відміну від змінних, значення констант може бути визначено тільки один раз, і далі не можна його змінити. Наприклад, визначимо числову константу:

define("NUMBER", 22);

Для визначення константи використовується **оператор *define***, який має таку форму:

define (string $name, string $value, bool $case_sen = false).

Параметр *$name* визначає назву константи, а параметр *$value* – її значення. Третій необов'язковий параметр приймає логічне значення: *true* або *false*. Якщо значення дорівнює *false*, то при використанні константи, буде враховуватися її регістр, якщо *true* – не враховується. У нашому випадку, третій параметр не використаний, тому, за замовчуванням, дорівнює *false*.

Після визначення константи, її можна використовувати так само, як і звичайну змінну. Головна відмінність – не можна змінити її значення. Інша відмінність від змінної – не потрібно використовувати знак $.

Крім створюваних програмістом констант, **у PHP є ще кілька вбудованих констант**:

- *__FILE__*: зберігає повний шлях і ім'я поточного файлу;
- *__LINE__*: зберігає поточний номер рядка, який обробляється інтерпретатором;
- *__DIR__*: зберігає каталог поточного файлу;
- *__FUNCTION__*: назва оброблюваної функції;
- *__CLASS__*: назва поточного класу;
- *__METHOD__*: назва оброблюваного методу;
- *__NAMESPACE__*: назва поточного простору імен.

7.1.5. Операції мови PHP

У PHP можна використовувати різні операції: арифметичні, логічні і т.д. Розглянемо кожен тип операцій.

1. Арифметичні операції:

- + (додавання);
- - (віднімання);
- * (множення);
- / (ділення);
- % (отримання цілочислової остачі від ділення);
- ++ (інкремент / збільшення значення на одиницю);
- -- (декремент / зменшення значення на одиницю).

Важливо розуміти відмінність між виразами *++$a* і *$a++* та *--$a* і *$a--*. Наприклад:

```
$a = 12;
$b = ++$a; // $b дорівнює 13
echo $b;
```

Спочатку до значення змінної *$a* додається одиниця, після чого її значення присвоюється змінній *$b*. Якщо записати вираз так: *$b = $a++;*, то спочатку значення змінної *$a* присвоється змінній *$b*, а потім відбудеться збільшення значення змінної *$a*.

2. Операції присвоєння:

- = – присвоєння змінній певного значення: *$a = 5;*
- += – додавання з подальшим присвоєнням результату, наприклад: *$a += 5;*
- -= – віднімання з подальшим присвоєнням результату;
- *= – множення з подальшим присвоєнням результату;
- /= – ділення з подальшим присвоєнням результату;

• .= – об’єднання рядків із подальшим присвоєнням результату, наприклад:

$b = "12";

$b. = "5";

echo $b; // 125

• %= – отримання цілочисельної остачі від ділення з подальшим присвоєнням результату.

3. Операції порівняння, як правило, використовуються в умовних конструкціях, коли потрібно порівняти два значення і в залежності від результату порівняння, виконати деякі дії. Є такі операції порівняння:

• == – оператор рівності порівнює два значення: якщо вони рівні, повертає *true*, інакше повертає *false*;

• === – оператор тотожності також порівнює два значення і типи змінних: якщо вони рівні, повертає *true*, інакше повертає *false*;

• != – оператор перевірки нерівності порівнює два значення: якщо вони не рівні, повертає *true*, інакше повертає *false*;

• !== – оператор перевірки тотожної нерівності порівнює два значення: якщо вони тотожно не рівні, повертає *true*, інакше повертає *false* (*$a!==5*);

• – оператор порівнює два значення: якщо перше більше другого, то повертає *true*, інакше повертає *false*;

• < – оператор порівнює два значення: якщо перше значення менше другого, то повертає *true*, інакше повертає *false* (*$a <5*);

• >= – оператор порівнює два значення: якщо перше більше або дорівнює другому, то повертає *true*, інакше повертає *false* (*$a>=5*);

• <= – оператор порівнює два значення: якщо перше менше або дорівнює другому, то повертає *true*, інакше повертає *false* (*$a<=5*);

Оператори рівності та тотожності порівнюють два вирази і повертають *true*, якщо вирази рівні. Але між ними є відмінність. Якщо в операції рівності приймають участь два значення різних типів, то вони приводяться до одного, який

вважається інтерпретатором, оптимальним. Наприклад:

$x = "22";

$y = 22;

if ($x == $y) echo "рівні";

else echo "не рівні";

Очевидно, що змінні зберігають однакові значення різних типів. При порівнянні вони будуть приводитися до одного типу – числового. Змінна *$x* буде перетворена до числа 22. У підсумку, обидві ці змінні виявляться рівними. Або, наприклад, такі змінні також будуть рівні:

$x = false; $y = 0;

Щоб уникнути подібних ситуацій використовується **операція тотожності (еквівалентності)**, яка враховує не тільки значення, але й типи змінних. У цьому випадку, порівняння розглянутих змінних дасть *false*.

4. Логічні операції, зазвичай, використовуються для об'єднання результатів двох операцій порівняння. Наприклад, потрібно виконати певну дію, якщо істинні кілька умов. Є такі логічні операції:

- **&&** – оператор повертає *true*, якщо обидві операції порівняння повертають *true*, інакше повертає *false*: *$a==5 && $b==6;*
- **||** – оператор повертає *true*, якщо хоча б одна операція порівняння повертає *true*, інакше повертає *false*: *$a == 5 || $b == 6;*
- **!** – оператор повертає *true*, якщо операція порівняння повертає *false*: *!($a>=5)*.
- ***xor*** – оператор повертає *true*, якщо тільки одне зі значень дорівнює *true*. Якщо обидві операції рівні *true* або жодна з них не дорівнює *true*, повертає *false*. Наприклад:

$x = 12;

$y = 6;

if ($x xor $y) echo 'true';

else echo 'false';

Результатом цієї логічної операції буде *false*, оскільки обидві змінні мають певне значення.

Аналогічно &&, ||, існують операції *and* та *or*.

5. Бітові операції виконуються порозрядно – над окремими бітами числа. При цьому числа розглядаються в двійковому поданні, наприклад, 2 в двійковій системі 010, число 7 – 111.

Логічне множення (&) виконується порозрядно: якщо у обох операндів значення розрядів дорівнює 1, то операція повертає 1, інакше – число 0. Наприклад:

$a1 = 4; // 100

$b1 = 5; // 101

echo $a1 & $b1; // дорівнює 4

Число 4 в двійковій системі являє собою 100, а число 5 – 101. Порозрядно помножимо числа і отримаємо (1×1, 0×0, 0×1) = 100, тобто число 4 в десятковій системі.

Логічне додавання (|) також виконується порозрядно: результуючий розряд дорівнює одиниці, якщо хоча б в одного числа, в даному розряді, 1.

Логічне заперечення (~) інвертує всі розряди: якщо значення розряду дорівнює 1, то стає рівним нулю, і навпаки.

Вираз *x<<y* зсуває число *x* вліво на *y* розрядів. Наприклад, *4<<1* зсуває число 4 (в двійковій системі 100) на один розряд вліво. Таким чином, отримуємо 1000 або число 8 в десятковій системі.

Вираз *x>>y* зсуває число *x* вправо на *y* розрядів. Наприклад, *16>>1* зсуває число 16 (в двійковій системі 10 000) на один розряд вправо. В результаті отримуємо 1000 або число 8 в десятковій системі.

Як відомо, для об'єднання рядків використовується **оператор "крапка"**. Якщо змінні не є рядками, тобто іншого типу, наприклад, числового, то їх значення перетворюються в рядки і також виконується операція об'єднання рядків.

7.1.6. Умовні конструкції та цикли

Умовні конструкції дозволяють направити роботу програми по одному з можливих шляхів у залежності від певної умови.

Конструкція *if (умова)* перевіряє істинність умови: якщо вона істинна, то виконується блок операторів, що розміщується після *if*; якщо умова хибна, тобто дорівнює *false*, то блок *if* не виконується. Наприклад:

$x = 4;

```
$y = 2;
if ($x> 0){
   $res = $x * $y;
   echo "результат: $res";  // 8
}
```

Блок операторів визначається **фігурними дужками**. В цьому випадку, умова істинна (тобто дорівнює *true*): значення змінної *$x* більше *0*, тому виконається блок операторів у фігурних дужках. Якби значення *$x* було б менше *0*, то блок *if* не виконувався б.

Якщо блок *if* містить всього один оператор, то фігурні дужки можна опускати. **Блок *else*** містить оператори, які виконуються, якщо умова після *if* хибна, тобто дорівнює *false*:

```
$x = 4;
$y = 2;
if ($x > 0){
   echo $x * $y;  // 8
} else {
   echo $x / $y;
}
```

Якщо *$x* більше *0*, то виконується блок *if*, якщо ні – блок *else*. Оскільки, в обох блоках по одному оператору, можна було опустити фігурні дужки при визначенні блоків. **Конструкція *elseif*** дозволяє задіяти додаткові умови в програмі:

```
$x = 5;
$y = 2;
if ($x <0)  echo $x * $y;
elseif ($x == 0) echo $x + $y;
elseif ($x == 5)  echo $x - $y;  // 3
else echo $x / $y;
```

Конструкцію можна доповнити довільною кількістю блоків *elseif*: якщо жодна з умов в *if* або *elseif* не виконується, то спрацьовує блок *else*.

Конструкція *switch..case* є альтернативою використання конструкції *if..elseif..else*. Наприклад:

```
$x = 1;
switch ($x) {
   case 1: echo "додавання"; break;  // додавання
```

```
    case 2: echo "віднімання"; break;
    case 3: echo "множення"; break;
    case 4: echo "розподіл"; break;
}
```

Після ключового слова *switch* в дужках вказується вираз для подальшого порівняння. Значення цього виразу послідовно порівнюється зі значеннями, розміщеними після операторів *case*: якщо збіг знайдено, то виконається відповідний блок *case*. В кінці блоку *case*, ставиться **оператор *break*,** щоб уникнути виконання інших блоків. Якщо ми хочемо обробити ситуацію, коли збіг не буде знайдено, то можна додати **блок *default***:

```
…
    default:
        echo "дія за замовчуванням";
        break;
}
```

Тернарна операція складається з трьох операндів і має наступне визначення:

[перший операнд – умова] ? [другий операнд]: [третій операнд];

Залежно від умови, тернарна операція повертає другий або третій операнд: якщо умова дорівнює *true*, то повертає другий операнд; якщо умова – *false*, то третій.

Для виконання повторюваних дій в PHP, як і в інших мовах програмування, використовуються цикли. У PHP є такі **види циклів**:

- *for;*
- *while;*
- *do..while*.

Цикл *for* має наступне формальне визначення:

```
for ([ініціалізація лічильника]; [умова]; [зміна лічильника]) {
    // дії
}
```

Розглянемо приклад циклу *for*:

```
for ($i = 1; $i <10; $i ++) {
    echo "Квадрат числа $i дорівнює ". $i * $i;
}
```

Перша частина оголошення циклу *$i = 1* створює та ініціалізує лічильник – змінну *$i*. Перед виконанням циклу її значення дорівнюватиме 1. Фактично це те ж саме, що й оголошення змінної.

Друга частина *$i <10* – умова, при якій буде виконуватися цикл. У цьому випадку цикл буде виконуватись, поки *$i* не досягне *10*.

Третя частина *$i ++*– приріст лічильника на одиницю. Необов'язково збільшувати лічильник на одиницю, можна, наприклад, зменшувати: *$i--*.

У результаті блок циклу спрацює 9 разів, поки значення *$i* не стане рівним 10. Кожного разу при виконанні циклу, це значення буде збільшуватися на 1. Кожне повторення циклу називається **ітерацією**. Таким чином, у цьому випадку, відбудеться 9 ітерацій.

Цикл *while* перевіряє істинність вказаної умови: якщо умова істинна, то виконується блок операторів циклу:

```
$i = 1;
while ($i <10) {
   echo $i * $i . "<br />";
   $i++;
}
```

Якщо в блоці *while* всього один оператор, то фігурні дужки блоку можна опустити.

Цикл *do..while* схожий на цикл while, але спочатку виконується блок циклу, а після – перевірка умови. Тобто, навіть якщо умова хибна, блок циклу виконається як мінімум один раз:

```
$i = 1;
do {
   echo $i * $i . "<br />";
   $i++;
}
while ($i < 10)
```

Іноді виникає ситуація, коли потрібно вийти з циклу, не чекаючи його завершення. У цьому випадку можна використовувати **оператор *break***:

```
for ($i = 1; $i <10; $i ++) {
```

```
    $res = $i * $i;
    if ($res > 80) break;
    echo "Квадрат числа $i дорівнює $res";
}
```

У прикладі вище вивід квадрату числа відбудеться тільки 8 разів, оскільки квадрат числа 9 більший за 80 і відбудеться вихід з циклу.

Для управління циклами також застосовується **оператор *continue***. Він здійснює перехід до наступної ітерації циклу:

```
for ($i = 1; $i <10; $i++) {
    if ($i == 5)  continue;
    echo "Квадрат числа $i дорівнює". $i * $i;
}
```

При виконанні програми, коли значення *$i* буде дорівнювати *5*, відбудеться перехід до наступної ітерації, при цьому всі інші оператори після оператора *continue* виконуватися не будуть.

7.1.7. Функції та область видимості змінних

Функція – це набір операторів, що виконують певну дію. Синтаксис визначення функції:

```
function імя_функції ([параметр [,...]]) {
    // оператори
}
```

Визначення функції починається з ключового слова *function*, після якого вказується ім'я функції. **Ім'я функції** має починатися з алфавітного символу або символу підкреслення, може складатися з будь-якої кількості алфавітно-цифрових символів або символів підкреслення.

Після імені функції в дужках перераховуються **параметри**. Якщо функція не має параметрів, дужки залишаються порожніми. Далі в фігурних дужках розміщується **тіло функції**, що являє собою набір операторів. Визначимо найпростішу функцію:

```
function display() { echo "Функція display()"; }
```

Ця функція не має параметрів і все, що вона робить – це виводить на сторінку вказане повідомлення. Щоб функція спрацювала, її потрібно викликати. Здійснимо виклик функції:

```
display();
function display() { echo "Функція display()"; }
```

Функція може **повертати деяке значення** – число, рядок і т.д. Для повернення значення у функції використовується **оператор *return***, після якого вказується значення, що повертається. Наприклад:

```
$result = get();
echo "Сума квадратів від 1 до 9 дорівнює $result"; // 285
function get() {
   $res = 0; // значення, що повертається
   for ($i = 1; $i < 10; $i++) { $res += $i * $i; }
   return $res;
}
```

Функція *get()* повертає число – суму квадратів від 1 до 9 включно. Це число зберігається в змінній *$res*. Завдяки оператору *return*, можна присвоїти значення, що повертає функція *get*, будь-якій змінній: *$result = get()*.

Створимо функцію з параметрами:

```
$result = get(1, 10);
echo "Сума квадратів від 1 до 9 дорівнює $result"; // 285
function get($lowlimit, $highlimit) {
   $res = 0; // значення, що повертається
   for ($i = $lowlimit; $i < $highlimit; $i++) $res += $i * $i;
   return $res;
}
```

Тепер функція *$get()* використовує параметри, тому потрібно при виклику цієї функції передати на місце параметрів деякі значення. Якщо при виклику не вказати значення для всіх параметрів, то в результаті отримуємо помилку:

```
$result = get(1); // Uncaught ArgumentCountError
```

Можна використовувати **для параметрів: значення, за замовчуванням**. Наприклад:

```
function get($lowlimit, $highlimit = 10) {
   $res = 0; // значення, що повертається
   for ($i = $lowlimit; $i <$highlimit; $i++) $res += $i * $i;
   return $res;
}
$result = get(1);
```

echo "Сума квадратів дорівнює $result"; // 285

У цьому випадку, якщо не вказати значення для другого параметра, то, за замовчуванням, воно буде дорівнювати 10.

Досі ми використовували передачу параметрів **за значенням**. Але в PHP є інша форма передачі параметрів – **за посиланням**. Розглянемо її і порівняємо ці два способи. Стандартна передача параметрів за значенням показана прикладами вище.

Розглянемо передачу параметрів за посиланням:

```
$num = 10;
get($num);
echo "\$num = $num";
function get(&$number){
   $number *= $number;
   echo "Квадрат дорівнює: $number";
}
```

Результат виконання коду:

Квадрат дорівнює: 100 $num = 100

При передачі за посиланням, перед параметром ставиться **знак амперсанда**: *function get(&$number)*. Тепер інтерпретатор буде передавати не значення змінної, а посилання на цю змінну в пам'яті!

При використанні змінних і функцій слід враховувати **області видимості змінних**. Область видимості визначає область дії та існування даної змінної. **Локальні змінні** створюються всередині функції, до таких змінних можна звернутися тільки в межах даної функції. Наприклад:

```
function get($lowlimit, $highlimit) {
   $res = 0; // значення, що повертається
   for ($i = $lowlimit; $i <$highlimit; $i++) $res += $i * $i;
   return $res;
}
$result = $res; // помилка, тому що $res – локальна змінна в функції
```

У цьому випадку в функції *get()* визначена локальна змінна *$res*. Із загального контексту не можна до неї звернутися, тобто записати *$result = $res*; тому що область дії змінної *$res* обмежена функцією *get()*. Поза цією функцією змінна *$res* не

існує. Те ж саме відноситься і до **параметрів функції:** поза функцією параметри *$lowlimit* і *$highlimit* також не існують. Як правило, локальні змінні зберігають якісь проміжні результати обчислень, як і в прикладі вище.

На локальні змінні схожі **статичні**. Вони відрізняються тим, що після завершення роботи функції, їх значення зберігається. При кожному новому виклику, функція використовує раніше збережене значення. Наприклад:

```
function getC() {
   static $c = 0; $c++; echo $c;
}
getC(); // c=1
getC(); // c=2
getC(); // c=3
```

Щоб вказати, що змінна буде статична, використовується **ключове слово *static***. При трьох послідовних викликах функції *getC()*змінна *$c* буде збільшуватися на одиницю. Якби змінна *$c* була звичайною – нестатичною, то при кожному виклику функція *getC()* виводила б 1. Як правило, статичні змінні служать для створення різних лічильників, як у прикладі вище.

Іноді потрібно, щоб **змінна була доступна всюди**, **тобто глобально**. Такі змінні можуть зберігати загальні для всієї програми дані. Для визначення глобальних змінних використовується **ключове слово *global***:

```
function getG() {
   global $g;
   $g = 20; echo $g;
  }
getG();
echo $g;
```

Після виклику функції *getG()* до змінної *$g* можна буде звернутися з будь-якої частини програми.

7.1.8. Підключення зовнішніх файлів

Якщо програма невелика, коду небагато, то всі операції можна визначити в одному файлі. Однак, як правило, програми містять велику кількість операцій. При визначенні всіх цих операцій в одному файлі, код може виглядати занадто

громіздким. Тому часто окремі частини коду розподіляють по окремих файлах, особливо коли ці частини можна використати в інших програмах на PHP.

Оператор *include* підключає до програми зовнішній файл з кодом php. Наприклад, визначимо **файл *factorial.php***:

```
function getFactorial($n) {
   $res = 1;
   for ($i = 1; $i <= $n; $i++) $res *= $i;
   return $res;
}
```

Тут відбувається обчислення факторіала. Тепер підключимо цей файл у нашу програму:

```
include "factorial.php";
$x = 5;
$f = getFactorial($x);
echo "Факторіал числа $x дорівнює $f"; //120
```

На місце визначення оператора *include* вставлятиметься весь код із файлу *factorial.php*. При цьому вставка файлу повинна відбуватися до використання функції, визначеної в цьому файлі.

Використання оператора *include* має недоліки. Наприклад, у різних місцях коду можна ненавмисно підключити один і той же файл, що при виконанні коду призведе до помилки. Щоб виключити повторне підключення файлу, замість оператора *include* використовують **оператор *include_once***:

```
include_once "factorial.php";
$x = 5;
$f = getFactorial($x);
echo "Факторіал числа $x дорівнює $f";
```

Тепер, якщо підключити цей же файл за допомогою *include_once* де-небудь нижче, то це підключення буде проігноровано, оскільки файл уже підключений до програми.

Дія **оператора *require*** подібна оператору *include*: він також підключає зовнішній файл, вставляючи в програму його вміст, але якщо файл не буде знайдено, дія програми припиниться. Також якщо в коді наявні кілька операторів *require*, які підключають один і той же файл, то інтерпретатор видасть помилку. Аналогічно, щоб уникнути такої ситуації, потрібно використовувати **оператор *require_once***.

7.1.9. Масиви

1. Визначення масиву. Ключі (індекси) елементів

Масиви призначені для зберігання наборів даних або елементів. Кожен елемент в масиві має свій унікальний ключ і значення. Отже, збережемо в масив список моделей телефонів:

```
$phones[0] = "Samsung Galaxy ACE II";
$phones[1] = "Nokia N9";
$phones[2] = "Samsung Galaxy III";
$phones[3] = "Sony Xperia Z3";
for ($i = 0; $i < count($phones); $i++)
   echo $phones[$i];
```

Вище створюється масив *$phones* з 4-ох елементів. **Кожен елемент у масиві** являє собою пару ключ-значення. Так, перший елемент $phones [0] = "Nokia N9" має ключ – число 0, а значення – рядок "Nokia N9". У таких масивах числові ключі ще називаються **індексами**. За допомогою **функції *count()*** можна отримати кількість елементів в масиві. Завдяки тому, що ключі нумеруються числами по порядку від 0 до 3, і знаючи розмір масиву, можна вивести елементи масиву в циклі *for*.

Щоб зрозуміти відношення ключів і значень елементів, можна вивести масив за допомогою **функції *print_r***:

```
print_r($phones);
```

Результат виглядає так:

```
Array ( [0] => Samsung Galaxy ACE II [1] => Nokia N9 [2] => Samsung Galaxy III [3] => Sony Xperia Z3 )
```

Якщо не вказується ключ елемента, то PHP в якості ключів використовує числа. При цьому нумерація ключів починається з нуля, а кожен новий ключ збільшується на одиницю. Знаючи ключ елемента в масиві, можна звернутися до цього елемента, отримати або змінити його значення. Але **ключами** можуть бути не тільки цілі числа, а й рядки (асоціативні масиви):

```
$phones["nokia"] = "Nokia N9";
$phones["sony"] = "Sony Xperia Z3";
$phones["samsumg"] = "Samsung Galaxy III";
$phones["apple"] = "iPhone5";
echo $phones["samsumg"];  // Samsung Galaxy III
```

Масиви, в яких ключі елементів являють собою рядки, називаються **асоціативними**.

2. Способи визначення масивів

Досі було розглянуто один спосіб створення масиву. Але є й інший, який передбачає використання **оператора *array()***:

$phones = array('Samsung Galaxy S III', 'iPhone', 'Nokia N9', 'Sony XPeria Z3');

Оператор *array()* приймає набір елементів. Тут також явно не вказуються ключі. Тому PHP, автоматично, нумерує елементи з нуля. Але можна вказати для кожного елемента ключ:

$phones = array("samsumg" => "Samsung Galaxy III", "apple" => "iPhone5", "Nokia" => "Nokia N9", "sony" => "Sony XPeria Z3");

echo $phones["samsumg"]; // Samsung Galaxy III

Операція => дозволяє зіставити ключ із певним значенням.

3. Цикл *foreach ... as*

Вище було розглянуто, як за допомогою циклу *for* вивести всі елементи масиву, де ключі задані числами від 0 до 3. Однак **з асоціативними масивами** це не працює. І для них в PHP призначений спеціальний тип циклу – ***foreach ... as***:

*foreach($phones as $item) echo "$item
";*

У циклі *foreach* послідовно зчитуються всі елементи масиву та їх значення, почергово, поміщаються в змінну, зазначену після ключового слова *as*. Тому змінна *$item*, почергово, містить кожне з чотирьох значень масиву *$phone*s. Коли буде зчитано останній елемент з масиву, цикл завершиться.

Цикл *foreach* дозволяє використовувати не тільки значення, а й ключі елементів:

foreach ($phones as $key => $value) echo $phones[$key];

При переборі елементів циклу в змінну *$key* буде передаватися ключ елемента, а в змінну *$value* – її значення.

Альтернативою циклу *foreach* є використання **функцій *list* і *each***:

while (list($key, $value) = each($phones))
* echo $phones[$key]; //або що те саме echo $value;*

Цикл *while* буде працювати, поки функція *each* не поверне значення *false*. Функція *each* проходить по всіх елементах масиву *$phones* і повертає поточний елемент у вигляді масиву, в який входять ключ і значення елемента. Потім цей масив

передається функції *list* і відбувається присвоєння значення масиву змінним всередині дужок. Коли функція *each* закінчить перебір елементів масиву *$phones*, вона поверне *false*, і дія циклу *while* буде завершена.

Проілюструємо дію функції *each*:

```
$mas = each($phones);
print_r($mas);
```

Вивід результату:

```
Array
(   [1] => Nokia N9
    [value] => Nokia N9
    [0] => nokia
    [key] => nokia
)
```

4. Багатовимірні масиви

У попередніх прикладах розглядалися тільки одновимірні масиви, де значення елементів – числа, рядки. Але в PHP масиви можуть також бути багатовимірними, тобто такими, де **елемент масиву сам є масивом**. Наприклад, створимо багатовимірний масив:

```
$phones = array (
"Samsumg" => array ("Galaxy VI", "Galaxy V"),
    "Apple" => array ("iPhone8", "iPhone9", "iPhone10"),
     "Nokia" => array("Nokia N9", "Nokia Lumia 930"),
     "Sony" => array("XPeria Z3", "Z3 Dual", "T2 Ultra"));
foreach($phones as $producer => $phone){
     echo "<h3> $producer </ h3>";
     echo "<ul>";
     foreach ($phone as $key => $value)
          echo "<li> $value </ li>";
     echo "</ul>";
}
```

Щоб звернутися до елемента цього масиву, також потрібно вказати ключі в квадратних дужках. Наприклад, звернемося до першого елементу в першому масиві. Оскільки ключ першого масиву – *"Apple"*, а ключ першого елемента в першому масиві – число 0 (тому що ключі явно не вказано), то

```
echo $phones["Apple"][0];  //iPhone8
```

Вкладені масиви також можуть бути асоціативними.

Функції для роботи з масивами:

- *is_array()* перевіряє, чи є змінна масивом, і якщо є, то повертає *true*, інакше – *false*;
- *count()* та *sizeof()* повертають кількість елементів масиву: *$number = count(назва масиву);*
- *shuffle()* перемішує елементи масиву випадковим чином;
- *compact* дозволяє створити з набору змінних асоціативний масив, де ключами будуть самі імена змінних.

5. У PHP є два **типи сортування**: сортування рядків за алфавітом і сортування чисел за зростанням/спаданням. Якщо сортовані значення є рядками, то сортування виконується за алфавітом, якщо числа – в порядку зростання чисел. PHP, за замовчуванням, самостійно вибирає тип сортування.

Для сортування за зростанням використовується **функція *asort***:

```
$tablets = array ("Samsung" => "Samsung Galaxy Tab 4",
                  "Lenovo" => "Lenovo IdeaTab A3500",
                  "Apple" => "Apple iPad Air");
asort($tablets);
echo "<ul>";
foreach ($tablets as $key => $tablet)
    echo "<li> $key: $tablet </li>";
echo "</ul>";
```

У цьому випадку значеннями масиву є рядки, тому PHP вибирає сортування за алфавітом. Однак за допомогою додаткового параметру можна явно вказати інтерпретатору PHP тип сортування. Цей параметр може набувати три значення:

- *SORT_REGULAR: автоматичний вибір сортування;*
- *SORT_NUMERIC: числове сортування;*
- *SORT_STRING: сортування за алфавітом.*

Щоб впорядкувати масив у зворотному порядку, використовується **функція *arsort***:

```
arsort($tablets);
```

Функція *asort* виконує сортування за значеннями елементів, але також існує і **сортування по ключам**, яке виконується **функцією *ksort()***. Сортування по ключам у зворотному порядку

виконується **функцією *krsort()***. Хоча вищеописані функції сортування чудово виконують свою роботу, але їх можливостей все ж недостатньо. Наприклад, відсортуємо за зростанням такий масив:

```
$operSystems=array("Windows 7","Windows 8","Windows 10");
asort($operSystems);
print_r($operSystems);
```

Результуючий вивід:

```
Array(
  [2] =>Windows 10
  [0]=>Windows 7
  [1]=>Windows 8
)
```

Оскільки значеннями елементів є рядки, то PHP сортує за алфавітом. Але таке сортування не враховує числа і регістр. Тому значення *"Windows 10"* буде на початку, а не в кінці, як повинно було бути. Для вирішення цієї проблеми в PHP є **функція *natsort()***, яка виконує природне сортування:

```
Array(
  [0] => Windows 7
  [1] => Windows 8
  [2] => Windows 10
)
```

Якщо потрібно ще при цьому, щоб сортування не враховувало регістр, то використовують **функцію *natcasesort()***.

7.1.10. Cookie

Cookie являють собою невеликі набори даних (не більше 4 кбайтів), за допомогою яких веб-сайт може зберегти на комп'ютері користувача будь-яку інформацію. За допомогою cookie можна відстежувати активність користувача на сайті: залогінений користувач чи ні, відстежувати історію його візитів і т.д. Для збереження cookie на комп'ютері користувача використовується **функція *setcookie()***, яка має таке визначення:

```
bool setcookie (string $name, string $value, int $expire,
string $path, string $domain, bool $secure, bool $httponly);
```

Параметри функції *setcookie()*:

• ***name***: ім'я cookie, яке буде використовуватися для доступу до його значення;

• ***value***: значення або вміст cookie – будь-який алфавітно-цифровий текст не більше 4 кбайт;

• ***expire*** (необов'язковий параметр): термін дії, після якого cookie знищується (якщо цей параметр не задано або дорівнює 0, то знищення cookie відбувається після закриття браузера);

• ***path*** (необов'язковий параметр): шлях до каталогу на сервері, для якого будуть доступні cookie. Якщо задати '/', cookie будуть доступні для всього сайту;

• ***domain*** (необов'язковий параметр): задає домен, для якого будуть доступні cookie. Якщо це домен другого рівня, наприклад *localhost.com*, то cookie доступні для всього сайту *localhost.com*, в тому числі і для його піддоменів типу *blog.localhost.com*;

• ***secure*** (необов'язковий параметр): вказує на те, що значення cookie має передаватися по протоколу HTTPS (якщо дорівнює *true*, cookie від клієнта буде передано на сервер, тільки якщо встановлено захищене з'єднання);

• ***httponly*** (необов'язковий параметр): якщо дорівнює *true*, cookie будуть доступні тільки через *http* протокол, тобто cookie, в цьому випадку, не будуть доступні скриптовою мовою, наприклад JavaScript.

Збережемо cookie:

```
$value1 = "Сінгапур";
$value2 = "китайська";
setcookie("city", $value1);
setcookie("lang", $value2, time() + 3600);
  // термін дії 1 година
```

Тут визначаються два елементи cookie: *"city"* і *"lang"*. Cookie *"city"* знищується після закриття браузера, а *"lang"* – через 3600 секунд, тобто через годину.

У cookie можна зберегти будь-яку інформацію, але не варто зберігати важливі з точки зору безпеки дані, наприклад паролі. А якщо і зберігати якусь важливу інформацію, то в зашифрованому вигляді.

Щоб **отримати cookie,** можна використовувати глобальний асоціативний масив **$_COOKIE**, наприклад *$_COOKIE["city"]*. Так можна отримати раніше збережені куки:

```
if (isset($_COOKIE["city"]))
    echo "Місто:".$_COOKIE ["city"]."<br>";
if (isset($_COOKIE["lang"]))
    echo "Мова:".$_COOKIE["lang"];
```

Щоб **видалити cookie** досить вказати для параметру *expire* (термін дії) – будь-який час у минулому:

```
setcookie("city", "", time() - 3600);
```

7.1.11. Сесії

Сесії – простий спосіб збереження інформації для окремих користувачів з унікальним ідентифікатором сесії. Фактично, це **набір змінних, які зберігаються на сервері** (або частина на сервері, а частина – в cookie браузера) і стосуються тільки активного користувача. Певною мірою сесії є альтернативою cookie в плані збереження даних про користувача.

Для запуску сесії необхідно викликати **функцію *session_start()***. Потім для збереження або отримання даних в сесії потрібно використовувати **глобальний асоціативний масив $_SESSION**. Збереження змінної в сесії:

```
$_SESSION["ім'я_змінної"] = $значення;
```

Отримання збереженого значення:

```
$змінна = $_SESSION["ім'я_змінної"];
```

Запустимо сесію і збережемо в ній значення:

```
session_start();
$_SESSION["city"] = "Сінгапур";
$_SESSION["lang"] = "китайська";
Тепер отримаємо ці значення і виведемо на сторінку:
if (isset($_SESSION["city"]) && isset($_SESSION["lang"])) {
   $city = $_SESSION["city"];
   $lang = $_SESSION["lang"];
   echo "Місто: $city <br /> Мова: $lang";
}
```

При запуску сесії за допомогою функції *session_start()*, якщо користувач перший раз заходить на сайт, PHP призначає йому **унікальний ідентифікатор сесії**, який, за замовчуванням,

називається "PHPSESSID". Цей ідентифікатор, за допомогою cookie, зберігається в браузері користувача. За допомогою спеціальних функцій можна його отримати:

```
session_start();
echo session_id(); // ідентифікатор сесії
echo session_name(); // ім'я – PHPSESSID
```

Те ж значення можна отримати, звернувшись до cookie безпосередньо:

```
echo $_COOKIE ["PHPSESSID"];
```

Сесія знищується із закриттям браузера, однак можна програмно закрити сесію за допомогою **функції *session_destroy()***:

```
session_start();
$_SESSION = array();
// знищення cookie з ідентифікатором сесії
if (session_id()!="" || isset($_COOKIE[session_name ()]))
   setcookie(session_name(),"", time() - 2592000, '/');
session_destroy();
```

При знищенні сесії, спочатку масиву $_SESSION присвоюється порожній масив, потім знищується пов'язані cookie PHPSESSID і далі викликається метод *session_destroy()*.

7.2. Робота з формами

7.2.1. Обробка форм. Метод POST

Одним із основних способів передачі даних між користувачем веб-сайту та сервером є обробка форм. За допомогою форм користувач може ввести дані й відправити їх на сервер. А сервер вже обробляє ці дані.

Форма являє собою спеціальний елемент розмітки HTML, який містять різні елементи вводу – текстові поля, кнопки і т.д. Створення форми складається з наступних кроків:

- створення елемента <*form*> </*form*> в розмітці HTML;
- додавання в цей елемент одного або декількох полів вводу;
- задання методу передачі даних: GET або POST;
- задання адреси, на яку будуть відправлятися введені

користувачем дані.

Створимо html-сторінку з формою. Визначимо **файл *form.php***, що міститиме наступний код:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "utf-8">
</head>
<body>
 <h3> Вхід на сайт </h3>
 <form action = "login.php" method = "POST">
    логін: <input type = "text" name = "login" /> <br>
    пароль: <input type = "text" name = "pass" /> <br>
    <input type = "submit" value = "Увійти">
 </form>
</body>
</html>
```

Атрибут *action = "login.php"* елемента *form* вказує, що дані з форми буде обробляти скрипт *login.php*, який перебуває з файлом *form.php* в одній папці. А **атрибут *method = "POST"*** вказує, що методом передачі даних буде метод POST.

Тепер створимо **файл *login.php*** із наступним вмістом:

```
<?php
  $login = "Невідомо";
  $pass = "Невідомо";
  if (isset($_POST["login"])) $login = $_POST["login"];
  if (isset($_POST["pass"])) $pass= $_POST["pass"];
  echo "Ваш логін: $login <br> Ваш пароль: $pass";
?>
```

Щоб отримати дані з форми, використовується **глобальна змінна $_POST**. Вона являє собою асоціативний масив даних, переданих за допомогою методу POST. Використовуючи ключі, можна отримати відправлені значення. Ключами в цьому масиві є значення атрибутів *name* полів вводу форми.

Наприклад, атрибут *name* поля вводу логіна має значення *login* (*<input type = "text" name = "login" />*), тому в масиві $_POST значення цього поля буде з ключем *"login"*: *$_POST["login"]*.

Оскільки можливі ситуації, коли поле вводу може бути не задано, наприклад, при прямому переході до скрипту: *…/login.php,* то бажано перед обробкою даних перевіряти їх наявність за допомогою **функції *isset()***. Якщо змінна визначена, то функція *isset()* повертає значення *true*.

При натисканні на кнопку відправки даних, дані, введені користувачем, методом POST будуть відправлені скрипту *login.php*. Необов'язково відправляти дані з форми іншому скрипту, можна обробити їх в тому ж файлі з формою *form.php*, доповнивши його відповідним кодом.

Велике значення в PHP має **організація безпеки даних**. Розглянемо декілька простих механізмів підвищення безпеки нашого веб-сайту. Візьмемо форму, яка наведена вище, і спробуємо ввести в неї деякі дані. Наприклад, введемо в поле для логіна "*<script>alert(hi);</script>*", а в поле для пароля – текст "*<h2>пароль</h2>*". Після відправки даних в html-розмітці буде код javascript, який виводить вікно з повідомленням "*hi*". Щоб уникнути подібних проблем з безпекою, рекомендується використовувати **функцію *htmlentities()***. Як параметр вона приймає значення, яке потрібно екранувати:

```
if (isset($_POST["login"]) && isset($_POST["pass"])) {
   $login = htmlentities($_POST["login"]);
   $pass = htmlentities($_POST["pass"]);
   echo "Ваш логін: $login <br> Ваш пароль: $pass";
}
```

Тепер після вводу коду html або javascript, усі теги будуть екрановані, javascript-код не виконається, а виведенням файлу *login.php* буде:

```
<script> alert(hi); </script>
<h2>пароль</h2>
```

Ще одна **функція *strip_tags()***, яка дозволяє повністю виключити теги html:

```
if (isset ($_POST["login"]) && isset ($_POST["pass"])) {
   $login = strip_tags($_POST["login"]);
   $pass = strip_tags($_POST["pass"]);
   echo "Ваш логін: $login <br> Ваш пароль: $pass";
}
```

Результатом її роботи при тому ж вводі буде такий вивід:
alert(hi);
пароль

7.2.2. Отримання даних. Метод GET

Іншим поширеним способом відправки даних на сервер є **метод GET.** Його суть полягає в тому, що дані передаються через адресний рядок браузера.

Щоб використовувати метод GET, потрібно при описі форми вказати атрибут ***method="get"***, тоді всі значення полів форми будуть передаватися через рядок запиту. Цей метод використовується формою, за замовчуванням.

Отже, створимо простий **скрипт *get.php*** наступного вмісту:

```
<?php
  $l = "не визначений";
  $a = "не визначений";
  if (isset ($_GET["login"]))  $l = $_GET["login"];
  if (isset ($_GET["age"]))  $a = $_GET["age"];
    echo "Ваш логін: $l <br> Ваш вік: $a";
?>
```

У PHP, за замовчуванням, визначено **глобальний асоціативний масив $_GET**, який зберігає всі значення, передані в рядку запиту. Такий масив подібний до масиву $_POST, оскільки, також за ключем можна отримати передане значення.

Тепер звернемося до цього скрипта через адресний рядок, наприклад, так:

http://localhost/get.php?login=mailcom&age=22

Якщо не передати значення для параметра, то відповідна змінна буде використовувати значення, по замовчуванню.

Усі параметри передаються на сервер у такій формі:

get.php?параметр1=значення1&параметр2=значення2&
параметрN=значенняN

Щоб передати список параметрів, після назви скрипта ставиться **знак питання**, за яким йдуть набори параметрів і їх значення. Назва параметра і буде ключем у масиві $_GET. Набори “параметр – значення” відокремлюються один від одного **знаком амперсанда** (&).

7.2.3. Отримання даних із нетекстових полів форми

Форми можуть містити різні елементи – текстові поля, прапорці, перемикачі і т.д., обробка яких має свої особливості.

Прапорці, або чекбокси (*☐*) можуть перебувати в двох станах: відміченому (*checked*) та невідміченому. Якщо прапорець знаходиться в невідміченому стані, то при відправці форми значення цього прапорця не передається на сервер. Якщо прапорець відмічений, то при відправці на сервер, наприклад, для поля *<input type="checkbox" name="remember"/>*, буде передано **значення *on***:

```
$remember = $_POST["remember"];
```

Якщо нас не влаштовує значення *on*, то за допомогою атрибута *value* можна встановити потрібне значення:

```
<input type="checkbox" name="remember" value="1"/>
```

Буває необхідно створити **набір чекбоксів**, де можна вибрати кілька значень. Наприклад:

```
<form>
  <h2>Форма вводу даних</h2>
  <form method="POST">
  ASP.NET:
  <input type = "checkbox" name = "technologies[]"
         value = "ASP" />
  PHP:
  <input type = "checkbox" name = "technologies[]"
        value = "PHP" />
  RUBY:
  <input type = "checkbox" name = "technologies[]"
          value = "Ruby" />
  <input type="submit" value="Відправити">
</form>
```

У цьому випадку, значення атрибуту *name* повинно мати квадратні дужки. Тоді після відправки сервер буде отримувати масив відмічених значень:

```
$technologies = $_POST["technologies"];
foreach ($technologies as $item) echo "$item <br />";
```

Тут змінна *$technologies* являє собою масив, який можна перебрати, і виконувати всі інші операції з масивами.

Перемикачі або радіокнопки дозволяють здійснити вибір між декількома взаємовиключними варіантами:

```
<form method="POST">
  <input type = "radio" name = "course" value = "ASP"/>
    ASP.NET
  <input type = "radio" name = "course" value = "PHP"/>
    PHP
  <input type = "radio" name = "course" value = "Ruby"/>
    RUBY
  <input type="submit" value="Відправити">
</form>
```

На сервер передається значення атрибута *value* вибраного перемикача. Отримання переданого значення:

```
if (isset($_POST["course"])) {
   $course = $_POST["course"]; echo $course;
}
```

Список реалізується елементом *select*, який дозволяє вибрати один або декілька елементів:

```
<form method="POST">
  <select name = "course" size = "1">
    <option value = "ASP"> ASP.NET </option>
    <option value = "PHP"> PHP </option>
    <option value = "Ruby"> RUBY </option>
    <option value = "Python"> Python </option>
  </select>
</form>
```

Елемент *select* містить набір варіантів вибору у вигляді елементів *option*. Отримати вибраний елемент можна наступним чином:

```
if (isset($_POST["course"])) {
   $course = $_POST["course"]; echo $course;
}
```

Елемент *select* також дозволяє множинний вибір. Такі списки мають атрибут *multiple*. При цьому обробка вибраних значень змінюється, оскільки сервер отримує масив значень:

```
<select name = "courses[]" size = "4" multiple = "multiple">
…
```

Для передачі масиву в атрибуті *name* також вказуються квадратні дужки: *name = "courses[]"*. Отримання даних здійснюється наступним чином:

```
if (isset($_POST["courses"])){
   $courses = $_POST["courses"];
   foreach ($courses as $item) echo "$item <br>";
}
```

7.2.4. Приклад обробки форми

Розглянемо комплексний приклад обробки форм, в якій об'єднаємо обробку різних елементів html. Визначимо таку **форму для вводу анкетних даних**:

```
<form action="input.php" method="POST">
   <p>Введіть ім'я:<br>
   <input type="text" name="firstname" /></p>
   <p>Форма навчання: <br>
     <input type="radio" name="eduform" value="очно"/>
       очна
     <input type="radio" name="eduform" value="заочно"/>
       заочна
   </p>
   <p>Необхідність гуртожитку:<br>
   <input type="checkbox"name="hostel" />Так</p>
   <p>Виберіть курси<br>
      <select name="courses[]" size="5" multiple="multiple">
         <option value="ASP.NET">ASP.NET</option>
         <option value="PHP">PHP</option>
         <option value="Ruby">RUBY</option>
         <option value="Python">Python</option>
         <option value="Java">Java</option>
      </select>
   </p>
   <p>Ваш коментар <br>
   <textarea name="comment" maxlength="200">
   </textarea></p>
   <input type="submit" value="Відправити">
</form>
```

Визначимо **скрипт *input.php***, який буде обробляти цю форму:

```
<?php
if (isset($_POST["firstname"]) && isset($_POST["eduform"])
  && isset($_POST["comment"])
  && isset($_POST["courses"])) {
    $name = htmlentities($_POST["firstname"]);
    $eduform = htmlentities($_POST["eduform"]);
    $hostel = "ні";
    if (isset($_POST["hostel"])) $hostel = "так";
    $comment = htmlentities($_POST["comment"]);
    $courses = $_POST["courses"];
    $output="<html>
        <head>
           <title>Анкетні дані</title>
        </head>
        <body> Ваше ім'я: $name<br/>
                Форма навчання: $eduform<br/>
                Необхідність гуртожитку: $hostel<br/>
                Вибрані курси: <ul>";
        foreach($courses as $item)
            $output.="<li>" . htmlentities($item) . "</li>";
        $output.="</ul></body></html>";
        echo $output;
     } else echo "Введені дані не коректні";
?>
```

Результатом виконання php-скрипта у файлі *input.php* буде веб-сторінка (рис. 7.4) при заповненій формі (рис. 7.3).

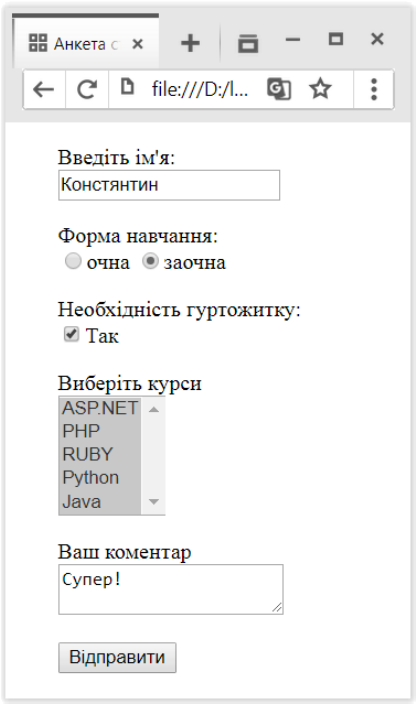


Рис. 7.3. Заповнена форма

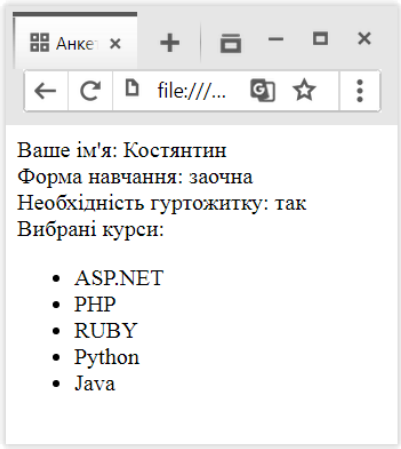


Рис. 7.4. Результат виконання скрипта *input.php*

7.3. Робота з базами даних MySQL

7.3.1. MySQL та phpMyAdmin

Як правило, як сховища даних використовуються бази даних. PHP дозволяє використовувати різні системи управління базами даних, але найбільш популярна на сьогодні у зв'язці з PHP є MySQL. MySQL – безкоштовне програмне забезпечення, що дозволяє взаємодіяти з базами даних за допомогою команд мови SQL.

Щоб спростити роботу з базами даних MySQL, існує спеціальний **набір скриптів phpMyAdmin**. phpMyAdmin являє собою інтуїтивний веб-інтерфейс для управління базами даних MySQL. Використовуючи цей інструмент, набагато легше працювати з базами даних, ніж керувати MySQL через консоль. При запуску OpenServer'а, phpMyAdmin доступний через його панель. Запустивши його, у лівій колонці можна побачити всі наявні бази даних на сервері MySQL. Навіть якщо ще не створено жодної бази даних, на сервері вже є деякий набір баз даних, за замовчуванням. У правій частині інтерфейсу phpMyAdmin містяться основні інструменти управління базами даних, а також різна конфігураційна інформація.

Щоб обмінюватися даними з сервером MySQL (зберігати, змінювати, видаляти, отримувати дані), потрібна база даних. **Створити базу даних** можна з консолі MySQL, а також з візуального інтерфейсу phpMyAdmin. Відкриємо інтерфейс phpMyAdmin. Перейдемо на вкладку «*Бази даних*». Під міткою «*Створити базу даних*» введемо ім'я нової БД, наприклад ***compstore***, і натиснемо на кнопку "*Створити*". Після цього отримаємо повідомлення про успішне створення нової БД, і вона буде додана в списки баз даних.

Нова база даних порожня і нічого не містить. **Додамо в неї таблицю** для зберігання даних. Для цього натиснемо на назві бази даних і перейдемо на вкладку «*Структура*», де визначаються опції нової таблиці. В поле "*Ім'я*" введемо назву нової таблиці. Нехай таблиця буде зберігати дані про моделі смартфонів, тому введемо назву "***phones***", і кількість стовпців – 3.

Для створення таблиці натиснемо на кнопку *"Вперед"*. Після цього потрібно задати **параметри стовпців**. Вкажемо, послідовно, назви стовпців: *id*, *name*, *company*. Задамо для кожного із них тип даних: для стовпця *id* – тип *INT*, для *name* і *company* – тип *VARCHAR*. Для стовпців *name* і *company* в полі «*Довжина/Значення*» вкажемо число 200 – максимальну довжину рядка в символах. Для стовпця *id* вкажемо в полі «*Індекс*» *PRIMARY*, а в полі «*A_I*» (*AutoIncrement*) поставимо галочку.

Таким чином, таблиця містить стовпці унікального ідентифікатора, назви телефону і назви виробника. Далі натиснемо внизу на кнопку "*Зберегти*". Після створення таблиці можна побачити в колонці баз даних таблицю і її стовпці.

Це не єдиний спосіб створення таблиць у phpMyAdmin, тому що тут можна **управляти базою даних за допомогою запитів SQL**. Отже, виділимо в списку баз даних нашу базу і перейдемо на вкладку «*SQL*». Вона відображає поле для вводу команди на мові запитів SQL. Введемо в нього наступну команду:

```
CREATE Table phones1 (
   id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR (200) NOT NULL,
   company VARCHAR (200) NOT NULL
)
```

Це стандартна команда створення таблиці на мові SQL. Після ключових слів *CREATE Table* вказується назва таблиці, далі в дужках через кому визначення стовпців. Кожне визначення стовпця містить назву стовпця, його тип і ряд додаткових значень. Наприклад, у визначенні стовпця *id:*

id INT NOT NULL AUTO_INCREMENT PRIMARY KEY

вказується його назва, тип – *INT*, а також те, що стовпець обов'язково повинен мати значення – *NOT NULL*, що його значення буде автоматично збільшуватися на одиницю з додаванням нового об'єкта – *AUTO_INCREMENT* і що він відіграє роль первинного ключа – *PRIMARY KEY*.

Створена в такий спосіб таблиця буде рівносильна тій, що була створена раніше. Натиснемо кнопку «*Вперед*», і **буде створена друга таблиця *phones1***.

7.3.2. Підключення до MySQL та виконання запитів

Для підключення до MySQL з PHP потрібно вказати **налаштування підключення**: адресу сервера, логін, пароль, назву бази даних і т.д. Оскільки зазвичай підключення до БД використовуються безліччю скриптів, то часто налаштування підключення виносять в окремий файл, завдяки чому легше їх оперативно змінювати. Отже, створимо **файл *connection.php*** і додамо в нього наступні рядки:

```
<?php
$host = 'localhost'; // адреса сервера
$database = 'compstore'; // назва бази даних
$user = 'root'; // ім'я користувача
$password = 'root'; // пароль
?>
```

Оскільки підключення до сервера буде проводитись на локальній машині, то адресою сервера буде *localhost*. За базу даних візьмемо створену базу даних *compstore*. За замовчуванням, на локальному сервері MySQL вже є користувач *root*, під яким будемо підключатися. Для нього необхідний пароль, який в OpenServer'і теж *root*. Після цього можна підключатися до бази даних:

```
require_once 'connection.php'; // підключаємо скрипт
  // підключаємося до сервера
$link = mysqli_connect($host, $user, $password, $database)
        or die("Помилка". mysqli_error ($link));
  // виконуємо операції з базою даних
mysqli_close($link); // закриваємо підключення
```

Для з'єднання застосовуємо **функцію *mysqli_connect()***, яка виконує всі конфігураційні налаштування і підключається до сервера. У разі помилки підключення спрацьовує **оператор *die()***, який виводить повідомлення про помилку й завершує роботу скрипта. При успішному підключенні функція *mysqli_connect()* повертає об'єкт підключення у вигляді **змінної *$link***. Після завершення роботи, підключення потрібно закрити. Для цього застосовується **функція *mysqli_close()***, яка має параметр – об'єкт підключення.

Щоб здійснити запит до бази даних, потрібно використовувати **функцію *mysqli_query()***, яка має два

параметри: об’єкт підключення і рядок запиту на мові SQL, наприклад:

```
// виконуємо операції з базою даних
$query = "SELECT * FROM phones";
$result = mysqli_query($link, $query) or die (mysqli_error($link));
if ($result) echo "Виконання запиту пройшло успішно";
```

Функція *mysqli_query()* повертає **об’єкт *$result***, який містить результат запиту. У разі невдачі цей об’єкт містить значення *false*.

7.3.3. Створення та видалення таблиць

Щоб **створити таблицю** потрібно використовувати вираз SQL "*CREATE TABLE*":

```
$query = "CREATE TABLE products (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR (200) NOT NULL,
    company VARCHAR (200) NOT NULL
) ";
$result = mysqli_query($link, $query) or die(mysqli_error($link));
if ($result) echo "Створення таблиці пройшло успішно";
```

Після виконання скрипта в базі даних буде створена таблиця *products*, яка за своїм визначенням буде аналогічна тим, що раніше створювалися через phpMyAdmin. Можна відкрити інтерфейс phpMyAdmin і побачити її.

Для видалення таблиці використовують вираз SQL "*DROP TABLE*":

```
$query = "DROP TABLE products";
$result = mysqli_query ($link, $query) or die(mysqli_error ($link));
if ($result) echo "Видалення таблиці пройшло успішно";
```

7.3.4. Додавання та отримання даних

Додамо дані в таблицю *products*, яка була створена вище. Для додавання даних використовується вираз “*INSERT*”:

```
$query = "INSERT INTO products VALUES
        (NULL, 'Samsung Galaxy III', 'Samsumg')";
```

Вираз *"INSERT"* вставляє в таблицю один рядок. Після ключового слова *INTO* вказується назва таблиці, а після *VALUES* в дужках перераховуються набір значень для всіх стовпців. Оскільки в таблиці три стовпці, то вказуються три значення. При створенні таблиці *products* вказано таку черговість стовпців: *id*, *name*, *company*, тому для стовпця *id* передається значення *NULL*, для *name – 'Samsung Galaxy III'*, а для *company – 'Samsumg'*. Оскільки стовпець *id* визначено як *AUTO_INCREMENT*, то необов'язково вказувати для нього певне числове значення, можна передати значення *NULL*, а MySQL присвоїть наступне доступне значення.

Розглянемо додавання даних на прикладі. Створимо **файл *create.php*** наступного вмісту:

```
…
<?php
require_once 'connection.php'; // підключаємо скрипт
if (isset ($_POST['name']) && isset($_POST ['company'])) {
  // підключаємося до сервера
 $link = mysqli_connect($host, $user, $password, $database)
 or die(mysqli_error($link))
  // екранування символів для mysql
 $n=htmlentities(mysqli_real_escape_string($link, $_POST['n']));
 $c=htmlentities(mysqli_real_escape_string($link, $_POST['c']));
  // створення рядка запиту
 $query = "INSERT INTO products VALUES(NULL, '$n', '$c')";
  // виконуємо запит
 $result=mysqli_query($link, $query) or die(mysqli_error($link));
 if ($result) echo "Дані додані";
  // закриваємо підключення
  mysqli_close($link);
}
?>
<h2> Додати новий товар </ h2>
<form method = "POST">
   <p> Модель: <br>
   <input type = "text" name = "n" /> </ p>
   <p> Виробник: <br>
   <input type = "text" name = "c" /> </ p>
```

```
    <input type = "submit" value = "Додати">
</form>
…
```

Вищенаведений код взаємодії з базою даних об'єднаний із функціональністю форми, оскільки з допомогою форми вводяться дані для запису в БД. Зокрема, використана **функція *mysqli_real_escape_string()*** для екранізації символів у рядку, який потім використовується в запиті SQL. Ця функція має два параметри: об'єкт підключення і рядок, який потрібно екранувати.

Таким чином, використовується екранізація символів, фактично, два рази: спочатку для sql-виразу за допомогою функції *mysqli_real_escape_string()*, а потім для html функцією *htmlentities()*. Це дозволяє захиститися одразу від двох видів атак: XSS-атак і SQL-ін'єкцій.

Для отримання даних використовується вираз sql "*SELECT*". Розглянемо приклад: створимо сторінку *index.php*, яка буде виводити значення з таблиці *products*.

```
…
$query = "SELECT * FROM products";
$result=mysqli_query($link,$query) or die (mysqli_error($link));
if ($result) {
        //кількість отриманих рядків
      $rows=mysqli_num_rows($result);
      echo "<table> <tr> <th> Id </ th> <th> Модель </ th>
            <th> Виробник </th> </tr>";
      for ($i = 0; $i < $rows; ++$i) {
        $row = mysqli_fetch_row($result);
        echo "<tr>";
        for ($j = 0; $j < 3; ++$j) echo "<td> $row[$j] </td>";
        echo "</ tr>";
      }
       echo "</table>";
      // очищаємо результат
      mysqli_free_result($result);
}
…
```

Для виведення результатів запиту, використовується цикл *for*. Для циклу *for* потрібно знати, скільки всього рядків отримано в змінній *$result*. Тому застосовується **функція *mysqli_num_rows()***.

Для проходу по рядках використовується такий цикл:

```
for ($i = 0; $i < $rows; ++$i) {
    $row = mysqli_fetch_row($result);
    echo "<tr>";
      for ($j = 0; $j < 3; ++$j) echo "<td> $row[$j] </td>";
    echo "</tr>";
}
```

Щоб отримати окремий рядок, використовується **функція *mysqli_fetch_row()***. Після виклику цієї функції вказівник у наборі *$result* переходить до наступного рядка, тому з кожним новим викликом отримується новий рядок.

Внутрішній цикл здійснює перебір значень поточного рядка. Оскільки при вибірці отримуються дані для всіх трьох стовпців таблиці, то лічильник *$j* проходить від 0 до 3. Кожен рядок являє собою масив комірок, тому за допомогою виразу *$row[$j]* можна отримати доступ до конкретної частини рядка.

Оскільки змінна *$result* після виконання запиту буде зберігати дані, то в кінці нам потрібно очистити пам'ять від непотрібних даних за допомогою **функції *mysqli_free_result()***.

Можна отримувати не всі дані, а, наприклад, дані для певних стовпців. Наприклад, отримаємо тільки назви моделей:

```
$query = "SELECT name FROM products";
$result=mysqli_query($link,$query) or die(mysqli_error($link));
if ($result) {
    echo "<ul>";
    while ($row = mysqli_fetch_row($result)) {
            echo "<li> $row[0] </ li>";
    }
    echo "</ ul>";
    mysqli_free_result($result);
}
```

У цьому випадку отримуємо дані тільки одного стовпця, тому кожен рядок у наборі *$result* міститиме тільки одну комірку, до якої звертаємося *$row[0]*.

7.3.5. Редагування та видалення даних

Редагування вже наявних у БД даних дещо складніше виконати, ніж отримання даних з БД. Перш за все тому, що потрібно поєднати отримання даних для редагування та оновлення бази даних новими значеннями.

Для оновлення застосовується вираз SQL "*UPDATE*":

```
$query = "UPDATE products SET name = 'Samsung ACE II',
          company = 'Samsung' WHERE id = '1'";
```

Після ключового слова **SET** перераховуються назви стовпців і нові значення для них. У кінці рядка запиту вказується селектор за допомогою **виразу *WHERE***. У даному випадку для рядка, у якого *id='1'*, задаються значення *name = 'Samsung ACE II'* і *company = 'Samsung'*.

Для видалення даних використовується вираз SQL "*DELETE*":

```
$query = "DELETE FROM products WHERE id = '5'";
```

У цьому випадку видаляється рядок, у якого *id = 5*. При видаленні, як і при редагуванні, потрібно передати скрипту *id* запису, який видаляється та виконати запит. Отже, створимо скрипт ***delete.php*** такого змісту:

```
<?php
require_once 'connection.php';
if (isset ($_GET ['id'])) {
  $link= mysqli_connect ($host, $user, $password, $database)
       or die("Помилка". mysqli_error($link));
  $id = mysqli_real_escape_string($link, $_GET['id']);
  $query = "DELETE FROM products WHERE id = '$id'";
  $result = mysqli_query($link, $query) or die ("Помилка".
    mysqli_error($link));
  mysqli_close($link);
}
?>
```

Звернемося до скрипту із запитом *…/delete.php?id = 1*, і скрипт видалить перший рядок у таблиці. Такий метод видалення простий і чудово працює. Але він небезпечний і рекомендується тільки як приклад видалення даних.

8. ТЕХНОЛОГІЯ AJAX

8.1. Загальні відомості

AJAX (Asynchronous JavaScript And XML – асинхронний JavaScript і XML) надає можливість асинхронно отримати контент з внутрішнього (локального) сервера без оновлення сторінки. Фактично AJAX не є новою технологією, але є новим підходом, який дозволяє оновлювати вміст веб-сторінки без перезавантаження.

Розглянемо приклад, щоб зрозуміти, як можна використовувати AJAX в повсякденній розробці додатків. Скажімо, потрібно створити сторінку, яка відображає інформацію профіля користувача з різними розділами, такими як особиста інформація, соціальна інформація, повідомлення і так далі.

Звичайний підхід полягає в створенні різних веб-сторінок для кожного розділу. Наприклад, користувач клацає по посиланню соціальної інформації, щоб перезавантажити браузер та відобразити цю сторінку. Це уповільнює навігацію між розділами, оскільки користувачеві доводиться кожного разу чекати перезавантаження браузера і повторного відображення сторінки.

З іншого боку, можна **використовувати AJAX** для створення інтерфейсу, який завантажує всю інформацію без оновлення сторінки. В цьому випадку можна відобразити вкладки для всіх розділів і, натиснувши на вкладку, він витягує відповідний вміст з внутрішнього сервера і оновлює сторінку без оновлення браузера. Це допомагає поліпшити загальне сприйняття кінцевого користувача.

Звичайний потік Ajax (рис. 8.1):

1. Користувач відкриває веб-сторінку синхронним запитом.

2. Користувач натискає на елемент DOM, зазвичай кнопку або посилання, яке ініціює асинхронний запит до сервера. Кінцевий користувач не помітить цього, оскільки виклик виконується асинхронно і не оновлює браузер. Однак можна розпізнати ці AJAX-виклики за допомогою такого інструменту, як Firebug.

3. У відповідь на запит AJAX сервер може повернути дані у формі XML, JSON або HTML.

4. Дані відповіді аналізуються з використанням JavaScript.

5. Проаналізовані дані оновлюються в DOM веб-сторінки.

Отже, можна бачити, що веб-сторінка оновлюється даними в реальному часі з сервера без перезавантаження веб-браузера.

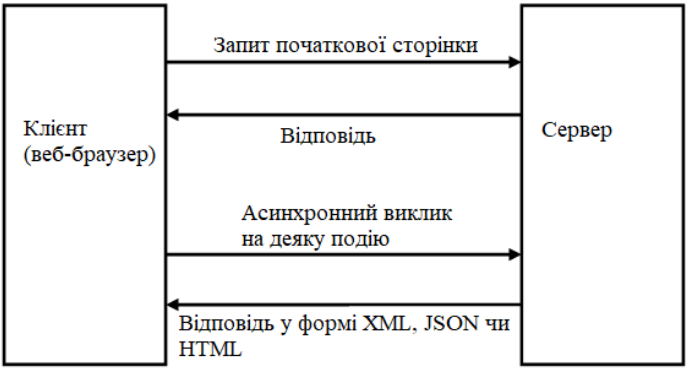


Рис. 8.1. Загальна схема роботи Ajax

8.2. AJAX та JavaScript

Розглянемо код JavaScript, який виконує виклик AJAX і асинхронно отримує відповідь від сервера:

```
<script>
var objXMLHttpRequest = new XMLHttpRequest();
objXMLHttpRequest.onreadystatechange = function() {
    if (objXMLHttpRequest.readyState === 4) {
        if (objXMLHttpRequest.status === 200) {
            alert(objXMLHttpRequest.responseText);
        } else {
        alert('Error Code:' + objXMLHttpRequest.status);
        alert('Error Message:' + objXMLHttpRequest.statusText);
        }
    }
}
objXMLHttpRequest.open('GET', 'request_ajax_data.php');
objXMLHttpRequest.send();
```

</script>

У коді спочатку відбувається ініціалізація **об'єкта XMLHttpRequest**, який відповідає за виконання викликів AJAX. Об'єкт XMLHttpRequest має **властивість *readyState***, значення якої змінюється протягом життєвого циклу запиту. Воно може містити одне з чотирьох значень: *OPENED, HEADERS_RECEIVED, LOADING* і *DONE*.

Можна налаштувати функцію-обробник для зміни стану, використовуючи **властивість *onreadystatechange***. Саме це зроблено в наведеному вище прикладі: використовується функція, яка буде викликатися кожного разу при зміні властивості стану.

У цій функції виконується перевірка на рівність значення *readyState* числу 4, що означає завершеність запиту та отримання відповіді від сервера. Далі перевіряється, чи дорівнює код стану числу 200, це означає, що запит був виконаний успішно. Після цього отримуємо відповідь, яка зберігається **у властивості *responseText* об'єкта XMLHttpRequest**.

Після налаштувань обробника, **ініціюється запит** викликом **методу *open*** об'єкта XMLHttpRequest. Після чого значення властивості *readyState* буде встановлено рівним 1. Наступним кроком є виклик **методу *send*** об'єкта XMLHttpRequest, який, фактично, відправляє запит на сервер. Після цього виклику значення властивості *readyState* буде дорівнювати 2.

Коли сервер відповість, він, в кінцевому підсумку, встановить значення *readyState* рівним 4. Після чого можна буде побачити вікно з попередженням, яке відображає відповідь сервера.

Таким чином, AJAX працює з JavaScript. Це був дуже простий приклад для демонстрації концепції AJAX, і в реальному додатку все може бути досить складно, оскільки потрібно обробити різні сценарії.

8.3. AJAX та jQuery

Бібліотека jQuery надає кілька різних методів для виконання викликів AJAX. Розглянемо **стандартний метод ajax**, який використовується найчастіше.

Розглянемо приклад:

```
<script>
$.ajax (
    'request_ajax_data.php', {
        success: function(data) {
          alert('AJAX call was successful!');
          alert('Data from the server' + data);
        },
        error: function() {
         alert('There was some error performing the AJAX call!');
        }
    }
);
</script>
```

Як відомо, **знак $** використовується для позначення об'єкта jQuery. **Першим параметром методу ajax** є URL-адреса, яка буде викликатися у фоновому режимі для отримання контенту зі сторони сервера. **Другий параметр** має формат JSON і дозволяє вказати значення для деяких параметрів, підтримуваних методом ajax. У більшості випадків, потрібно вказувати функції зворотного виклику для успішного виконання та помилки. Функція зворотного виклику для успішного виконання буде викликана після вдалого завершення виклику AJAX. При цьому відповідь, повернута сервером, буде передана цій функції. З іншого боку, функція зворотного виклику для помилки буде викликана, якщо щось піде не так і виникне проблема при виконанні виклику AJAX.

Отже, як можна побачити, AJAX-операції легко виконувати за допомогою бібліотеки jQuery. Фактично процес більш-менш однаковий, незалежно від бібліотеки JavaScript, яка використовується для викликів AJAX.

8.4. AJAX та PHP

Розглянемо код, який витягує контент JSON із файлу PHP на стороні сервера, використовуючи AJAX. Для демонстрації створимо приклад, який виконує вхід користувача в систему із використанням AJAX і jQuery. Створимо **файл *index.php***, як показано в наступному фрагменті, який відображає базову форму входу.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://code.jquery.com/jquery-3.3.1.js"
   integrity="sha256-
      2Kok7MbOyxpgUVvAk/HJ2jigOSYS2auK4Pfzbm7uH60="
   crossorigin="anonymous"></script>
</head>
<body>
<form id="loginform" method="post">
  <div>
   Username:
   <input type="text" name="username" id="username" />
   Password:
   <input type="password" name="password" id="password"/>
   <input type="submit" name="loginBtn" id="loginBtn"
        value="Login" />
  </div>
</form>
<script type="text/javascript">
$(document).ready(function() {
    $('#loginform').submit(function(e) {
        e.preventDefault();
        $.ajax({
            type: "POST",
            url: 'login.php',
            data: $(this).serialize(),
            success: function(response) {
                var jsonData = JSON.parse(response);
                // user is logged in successfully in the back-end
```

```
                // let's redirect
                if (jsonData.success == "1") {
                    location.href = 'my_profile.php';
                } else {
                    alert('Invalid Credentials!');
                }
            }
        });
    });
});
</script>
</body>
</html>
```

Файл *index.php* містить стандартну HTML-форму з полями для вводу імені користувача та пароля, а також фрагмент JavaScript jQuery. В прикладі використовується **подія *submit* елемента форми**, яка спрацьовує, коли користувач натискає кнопку відправки. В цьому обробнику подій ми ініціювали виклик AJAX, який асинхронно відправляє дані з форми в файл *login.php*, використовуючи метод POST. Отримавши відповідь від сервера, ми аналізуємо її, використовуючи **метод *parse* об'єкта JSON**. І нарешті, в залежності від успішного чи невдалого результату, виконуємо певні дії.

При цьому **файл *login.php*** виглядає так:

```
<?php
if (isset($_POST['username']) && $_POST['username'] &&
isset($_POST['password']) && $_POST['password']) {
    // do user authentication as per your requirements
    // ...
    // based on successful authentication
    echo json_encode(array('success' => 1));
} else {
    echo json_encode(array('success' => 0));
}
?>
```

Файл *login.php* містить логіку аутентифікації користувача і повертає відповідь JSON, яка базується на успішному або невдалому вході в систему.

9. ЗАВДАННЯ ДЛЯ ЛАБОРАТОРНИХ РОБІТ

Виконані завдання з курсу «Веб-програмування та веб-дизайн» будуть оцінюватись за такими параметрами:

1. Функціональність веб-сторінок згідно з вимогами завдання.
2. Розмір веб-сайту: кількість сторінок та об'єм контенту.
3. Дизайн веб-сторінок та його адаптивність.

Для виконання лабораторних завдань потрібно: текстовий редактор (Notepad++), веб-браузер, локальний або віддалений веб-сервер з інтерпретатором PHP та СКБД MySQL(OpenServer, …)

Всі лабораторні завдання курсу виконуються над одним об'єктом, який при виконанні всіх або частини завдань стає повноцінним веб-сайтом.

Не всі лабораторні завдання тісно пов'язані між собою, що дає змогу студентам виконувати не всі завдання, а лише ті, для виконання яких мають достатній рівень знань. Обов'язковими лабораторними завданнями можна вважати тільки перші два, оскільки без їх виконання неможлива подальша робота над іншими завданнями.

9.1. Розробка статичного веб-сайту

Розробити статичний веб-сайт, використовуючи мову розмітки HTML. Веб-сайт має складатися із, як мінімум, трьох сторінок: головна (стартова) сторінка сайту, сторінка з детальною інформацією про розробника (автора) та сторінка з формою зворотного зв'язку. Сторінки сайту повинні бути взаємопов'язані гіперпосиланнями. Тему для змістовного наповнення стартової сторінки автор обирає самостійно.

Стартова сторінка повинна мати семантичну структуру, тобто складатися із шапки сайту (*header*) з навігацією (*nav*), основної частини (*main*), підвалу (*footer*) з контактними даними та бокове меню.

Сайт повинен містити:

- заголовки;
- нумеровані та ненумеровані списки (маркери мають бути

різними);
- таблицю;
- фрейм, який завантажує ваше улюблене відео або сайт;
- посилання на абзаци сторінки;
- форму відправки повідомлення зворотного зв'язку з html-валідацією.

9.2. Додавання css-стилів до створеного веб-сайту

1. Для створеного, в попередній лабораторній роботі, статичного веб-сайту розробити css-стилі, в яких використати:
   - селектори нащадків;
   - селектори дочірніх елементів;
   - селектори елементів одного рівня;
   - псевдокласи;
   - псевдокласи дочірніх елементів;
   - псевдокласи форм;
   - псевдоелементи;
   - селектори атрибутів;
   - спадкування стилів;
   - каскадність стилів.

   При розробці css-коду стилізувати шрифти, абзаци, списки, таблицю.

2. Задати внутрішні та зовнішні відступи, границі, тінь елементів, лінійний та радіальний градієнт.

3. Задати для деяких елементів абсолютне, фіксоване та відносне позиціонування.

4. Розробити для веб-сторінок блокову верстку з використанням властивості *float*.

5. Розробити не складні анімації для наявних на сторінці елементів.

9.3. Модулі CSS3: FlexBox та Grid Layout

Створити адаптивну верстку Вашого веб-сайту, тобто розробити css-стилі для коректного відображення на планшетних та мобільних пристроях. Для верстки використати один з модулів CSS3: FlexBox або GridLayout. Обраний підхід має бути використаний для кількох сторінок. Тому при

необхідності створити додаткові веб-сторінки зі змістовним контентом для демонстрації результатів роботи.

9.4. Фреймворк Bootstrap

Підключити фреймворк Bootstrap до Вашого веб-сайту та використовувати його можливості. Для цього створити адаптивну верстку декількох новостворених веб-сторінок. При розробці останніх, використати можливості фреймворку у створенні:

- форм та кнопок (forms, inputs);
- вкладок (tabs);
- каруселі (carousel);
- спливаючих підказок (tooltips);
- таблиць (tables);
- зображень (images);
- пагінації (pagination);
- хлібних крихт (breadcrumbs);
- колапсу (collapse);
- випадаючого меню (dropdown);
- модальних вікон (modals);
- фільтрів (filters).

9.5. JS-валідація html-форми

Для створених html-форм на розробленому веб-сайті застосувати мову програмування JavaScript для перевірки полів форми на коректність введених даних.

Створена форма повинна містити: поля для вводу прізвища та імені, паспортних даних, ІПН, телефону, електронної пошти, адреси та інше. Серед них визначити поля, в які можна вводити тільки числа, текст або ж інші дані конкретного типу. У випадку коректності введених даних, виводити повідомлення про успішне заповнення форми при натисканні командної кнопки. А у випадку не коректності даних, виводити відповідне повідомлення та виділяти поля з помилками.

9.6. JS онлайн-калькулятор

Створити онлайн-калькулятор обчислення суми для вибраних значень елементів форми та деякому вмістимому вашої сторінки. На формі повинні бути використані такі елементи:

- *radio*;
- *checkbox*;
- *select*;
- *select* з множинним вибором;
- *input*;
- *textarea*.

Калькулятор повинен реагувати на будь-яку зміну елементів підрахунку, тобто при зміні значення елемента *select*, повинна переобчислюватися загальна сума.

Наприклад, онлайн-калькулятор сумарної вартості обраних пунктів меню вашої улюбленої піцерії чи ресторану або калькулятор обчислення вартості певного товару з врахуванням чи без врахування деяких його опцій.

9.7. Пошук на веб-сторінці засобами JS

Створити форму, що міститиме поле для вводу шуканого слова/фрази та кнопку запуску пошуку інформації на веб-сторінці. Натискання кнопки запускає процес пошуку, результатом якого має бути:

- кількість знайдених співпадінь та їх виділення зеленим кольором у випадках повного збігу;
- кількість знайдених співпадінь та їх виділення жовтим кольором у випадках часткового збігу, тобто знайдена інформація є словоформами шуканого тексту.

Наприклад, у результаті пошуку фрази ”гарна дівчина” в тексті буде виділено зеленим кольором всі входження цього словосполучення та жовтим всі входження таких фраз: ”гарної дівчини”, ”гарну дівчину”, ”гарній дівчині”. Але фраза ”гарнесенька дівчинонька” повинна залишатися поза увагою та не виділятися жовтим кольором, оскільки процент схожості менше 50%.

9.8. Плагін jQuery

Використати можливості плагіну jQuery на Ваших веб-сторінках:

1. Завантажити в директорію з веб-сайтом плагін jQuery.
2. Підключити його до розробленого веб-сайту.
3. Використати можливості плагіну для зміни css-стилів, додавання класів, видалення атрибутів елементів DOM, отримання вмістимого певного html-елемента (наприклад, абзацу <*p*>).
4. Створити анімацію відображення та приховання інформації на веб-сторінці при настанні певної події миші чи клавіатури.

9.9. Авторизація доступу до сайту (PHP)

Розробити сторінку авторизації для доступу до веб-сайту. Для цього потрібно:

- встановити OpenServer, або іншу збірку для роботи сервера Apache+PHP+MySQL;
- розмістити розроблений веб-сайт на сервері та запустити його через URL-адресу;
- створити у базі таблицю для зберігання даних користувачів (логін, пароль) з ролями (адміністратор, користувач);
- внести в таблицю свій логін та пароль як адміністратора сайту;
- створити сторінку авторизації через логін та пароль;
- розробити механізм авторизації звичайного користувача, що надає доступ до сторінок сайту;
- розробити механізм авторизації адміністратора, що надає доступ до веб-сайту з додатковою сторінкою (адмін-панель);
- створити додаткову перевірку при переході в адмін-панель на роль адміна, що дасть можливість уникнути випадку відображення звичайному користувачу адмін-панелі за url-адресою;
- для розробки механізму авторизації використати сесії та cookies.

9.10. Запис даних із html-форми в БД (PHP)

Створити таблиці в базі даних для зберігання даних, які відправляються формами веб-сайту. Написати програмний код мовою PHP для збереження та отримання, відправленої з форм, інформації в таблицю/таблиці бази даних (БД).

При додаванні даних в БД здійснювати перевірку на наявність в базі даних, введеної на формі інформації. Наприклад, перевірка на унікальність електронної адреси, номера телефону, паспортних даних, ідентифікаційного коду за наявності таких полів у html-формах.

9.11. Адмін-панель на PHP

Розробити адмін-панель з можливостями:

- перегляду, редагування, видалення записів в таблиці створеної БД;
- додавання нового запису в таблицю;
- пошуку відповідних записів по певному полю;
- додавання нових користувачів.

9.12. Відправка даних форми за допомогою технології Ajax

Використовуючи мову програмування JavaScript та технологію Ajax, реалізувати відправку значень полів форми. Відправку значень здійснити у форматі JSON. Засобами мов PHP та SQL виконати запис у відповідну таблицю бази даних MySQL.

10. ЕЛЕМЕНТИ КОНТРОЛЮ

10.1. Контрольні питання

1. Основи HTML5.
2. Елементи HTML5.
3. Елементи HTML-форм.
4. Основи CSS3.
5. Селектори CSS3.
6. Властивості CSS3.
7. CSS3: позиціонування елементів на сторінках.
8. Блокова верстка.
9. Трансформації, переходи, анімації в CSS3.
10. Технологія верстки Grid Layout.
11. Технологія верстки Flexbox.
12. Система та параметри сітки Bootstrap 4.
13. Основні класи Bootstrap 4 для роботи з кольором, таблицями, списками, зображеннями, сповіщеннями, кнопками.
14. Основні класи Bootstrap 4 для роботи з хлібними крихтами, пагінацією, меню, вкладками.
15. Bootstrap 4: модальні вікна, карусель, акордеон.
16. Способи підключенння js-скриптів до html.
17. Змінні, константи та типи даних в JavaScript.
18. Операції JavaScript.
19. Масиви в JavaScript.
20. Цикли в JavaScript.
21. Умовні конструкції в JavaScript.
22. Функції та область видимості змінних в JavaScript.
23. Замикання в JavaScript.
24. Самовикликаючі, рекурсивні та стрілочні функції в JavaScript.
25. Перевизначення функцій в JavaScript.
26. Передача параметрів за значенням та за посиланням в JavaScript.
27. Об’єкти та масиви в JavaScript.
28. Перевірка існування методів та властивостей в JavaScript.
29. Конструктори об’єктів у JavaScript.

30. Прототипи в JavaScript.
31. Інкапсуляція в JavaScript.
32. Успадкування в JavaScript.
33. Класи в JavaScript.

10.2. Тести по CSS3, HTML5

1. Елемент <*p*> – це тег, який …

а) містить наповнення, розташовується з нового рядка, і фактично, визначає новий абзац;

б) створює блок, який, за замовчуванням, розтягується по всій ширині браузера, а наступний після нього елемент переноситься на новий рядок;

в) виводить блок попередньо відформатованого тексту; такий текст відображається зазвичай моноширинним шрифтом і з усіма пробілами між словами;

г) призначений, переважно, для стилізації вкладеного в нього тексту, а саме дозволяє виділити частину інформації всередині інших тегів і встановити для неї свій стиль.

2. Елемент <*div*> – це тег, який …

а) містить наповнення, розташовується з нового рядка, і фактично, визначає новий абзац;

б) створює блок, який, за замовчуванням, розтягується по всій ширині браузера, а наступний після нього елемент переноситься на новий рядок;

в) виводить блок попередньо відформатованого тексту; такий текст відображається зазвичай моноширинним шрифтом і з усіма пробілами між словами;

г) призначений, переважно, для стилізації вкладеного в нього тексту, а саме дозволяє виділити частину інформації всередині інших тегів і встановити для неї свій стиль.

3. Елемент <*pre*> – це тег, який …

а) містить наповнення, розташовується з нового рядка, і фактично, визначає новий абзац;

б) створює блок, який, за замовчуванням, розтягується по всій ширині браузера, а наступний після нього елемент переноситься на новий рядок;
в) виводить блок попередньо відформатованого тексту; такий текст відображається зазвичай моноширинним шрифтом і з усіма пробілами між словами;
г) призначений, переважно, для стилізації вкладеного в нього тексту, а саме дозволяє виділити частину інформації всередині інших тегів і встановити для неї свій стиль.

4. Елемент <*span*> – це тег, який …

а) містить наповнення, розташовується з нового рядка, і фактично, визначає новий абзац;
б) створює блок, який, за замовчуванням, розтягується по всій ширині браузера, а наступний після нього елемент переноситься на новий рядок;
в) виводить блок попередньо відформатованого тексту; такий текст відображається зазвичай моноширинним шрифтом і з усіма пробілами між словами;
г) призначений, переважно, для стилізації вкладеного в нього тексту, а саме дозволяє виділити частину інформації всередині інших тегів і встановити для неї свій стиль.

5. Для створення маркованих списків використовуються теги …

а) , <li>;
б) , <li>;
в) , <ol>;
г) , <ol>, <li>.

6. Для створення нумерованих списків використовуються теги …

а) , <li>;
б) , <li>;
в) , <ol>;
г) , <ol>, <li>.

7. Значення *_blank* атрибута посилань (<*a* …> … </*a*>) *target* відповідає за …

а) відкриття html-документа в новому вікні або вкладці браузера;

б) відкриття html-документа в тому ж фреймі (або вікні);

в) відкриття документа у батьківському фреймі, якщо посилання розташовано у внутрішньому фреймі;

г) відкриття html-документа на все вікно браузера.

8. Значення *_self* атрибута посилань (<*a* …> … </*a*>) *target* відповідає за …

а) відкриття html-документа в новому вікні або вкладці браузера;

б) відкриття html-документа в тому ж фреймі (або вікні);

в) відкриття документа у батьківському фреймі, якщо посилання розташовано у внутрішньому фреймі;

г) відкриття html-документа на все вікно браузера.

9. Елемент *footer* …

а) являє собою вступний елемент, що передує основному вмісту, де можуть бути заголовки, елементи навігації або будь-які інші допоміжні елементи, наприклад, логотип, форма пошуку і т.п. ;

б) задає нижній колонтитул документу або розділу; зазвичай, містить інформацію про те, хто автор контенту на веб-сторінці, копірайт, дату публікації, блок посилань на схожі ресурси і т.д.;

в) об'єднує пов'язані між собою частини *html*-документа, виконуючи їх групування;

г) містить набір посилань, як правило, у вигляді ненумерованого списку, що є навігацією по сайту.

10. Елемент *header* …

а) являє собою вступний елемент, що передує основному вмісту, де можуть бути заголовки, елементи навігації або будь-які інші допоміжні елементи,

наприклад, логотип, форма пошуку і т.п. ;

б) задає нижній колонтитул документу або розділу; зазвичай, містить інформацію про те, хто автор контенту на веб-сторінці, копірайт, дату публікації, блок посилань на схожі ресурси і т.д. ;

в) об'єднує пов'язані між собою частини *html*-документа, виконуючи їх групування;

г) містить набір посилань, як правило, у вигляді ненумерованого списку, що є навігацією по сайту.

11. Значення *radio* атрибуту *type* поля *input* для вводу даних HTML-форми задає …

а) радіокнопку або перемикач (з групи радіокнопок можна вибрати тільки одну) ;

б) елемент прапорець, який може знаходитися у відміченому або невідміченому стані;

в) кнопку відправки даних з форми;

г) немає такого значення атрибуту *type*.

12. Значення *checkbox* атрибуту *type* поля *input* для вводу даних HTML-форми задає …

а) радіокнопку або перемикач (з групи радіокнопок можна вибрати тільки одну) ;

б) елемент прапорець, який може знаходитися у відміченому або невідміченому стані;

в) кнопку відправки даних з форми;

г) немає такого значення атрибуту *type*.

13. Крім елемента *input* у різних модифікаціях на формі також можна використовувати такі елементи:

а) button, textbox, label, select;

б) button, textarea, label, listbox;

в) button, textbox, label, listbox;

г) button, textarea, label, select.

14. Атрибут *pattern* елемента *input (type=text)* задає …

а) напрямок тексту;

б) максимально допустиму кількість символів у

текстовому полі;

в) шаблон, якому повинен відповідати текст, що вводиться;

г) текст-підказку для вводу.

15. Атрибут *placeholder* елемента *input (type=text)* задає …

а) напрямок тексту;

б) максимально допустиму кількість символів у текстовому полі;

в) шаблон, якому повинен відповідати текст, що вводиться;

г) текст-підказку для вводу.

16. Які значення за допомогою нижченаведеного *input'а* можна відправити на сервер?

<input type="number" list="priceList" step="10" min="30" max="70" value="50" id="price" name="price"/>

а) 10, 20, 30;

б) 30, 40, 50, 60, 70;

в) 30, 40, 50;

г) будь-які значення.

17. Які значення за допомогою нижченаведеного *input'а* можна відправити на сервер?

<input type="text" placeholder="+X-XXX-XXX-XXX" pattern="\+\d-\d{3}-\d{3}-\d{4}" id="phone" name="phone"/>

а) +X-XXX-XXX-XXX, +X-XXX-XXX-XXXX, де X – цифра;

б) +X-XXX-XXX-XXX, де X –цифра;

в) будь-які значення;

г) +X-XXX-XXX-XXXX, де X –цифра.

18. Які значення за допомогою нижченаведених *input'ів* можна відправити на сервер?

<input type="radio" value="HTML" checked name="tech"/>HTML
<input type="radio" value="JS" name="tech2"/>JavaScript

а) тільки HTML;

б) HTML, JS;

в) HTML, JavaScript;
г) тільки JS.

19. Які значення за допомогою нижченаведених *input'ів* можна відправити на сервер?

<input type="checkbox" value="Csharp" name="tech"/>C#
<input type="checkbox" value="Java" name="tech"/>Java

а). тільки Csharp;
б) C#, Java;
в) Csharp, Java;
г) тільки Java.

20. Значення CSS-властивості *background-color* для визначення кольору фону HTML-елемента можна задати …

а) значеннями *#RRGGBB*, де *RR*, *GG*, *BB* – шістнадцяткові числа;
б) назвою кольору англійською мовою для певної сукупності найпоширеніших кольорів;
в) значеннями *rgb(X, X, X)*, *rgba(X, X, X, D)*, де *X* є 0..255, *D* є 0..1;
г) будь-якими варіантами, переліченими у всіх відповідях.

21. Найвищий пріоритет мають …

а) вбудовані стилі (inline-стилі) ;
б) стилі, визначені в елементі *style* в тому ж файлі;
в) стилі з підключеного зовнішнього файлу;
г) стилі, визначені в елементі *script*.

22. CSS-стилі HTML-елементів можна задати з використанням …

а) селектора класу, ідентифікатора елемента, об'єкта класу;
б) селектора тегу, універсального селектора, селектора класу, атрибутів HTML-тегів;
в) селектора тегу, універсального селектора, селектора класу, ідентифікатора елементу;

г) поля класу, універсального селектора, селектора класу, ідентифікатора елементу.

23. Задати стилі для всіх елементів *span*, які розміщені після *div* на одному рівні вкладеності, можна, використовуючи селектор …

а) div span;
б) div>span;
в) div~span;
г) div+span.

24. Задати стилі для елементів *span*, які розміщені в елементі *div* на першому рівні вкладеності, можна, використовуючи селектор …

а) div span;
б) div>span;
в) div~span;
г) div+span.

25. Задати стилі для елемента *span*, який розміщений безпосередньо після *div*, можна, використовуючи селектор …

а) div span;
б) div>span;
в) div~span;
г) div+span.

26. Задати стилі для посилання в момент переходу по ньому можна з використанням псевдокласу …

а) :active;
б) :focus;
в) :visited;
г) :hover.

27. Задати стилі для елемента в момент наведення на нього курсору миші можна з використанням псевдокласу …

а) :active;
б) :focus;
в) :visited;

г) :hover.

28. Для додавання повідомлення після HTML-елемента використовується псевдоелемент …

а) ::before;
б) ::after;
в) ::first-line;
г) ::selection.

29. Для застосування стилів до виділеного користувачем фрагмента тексту використовується псевдоелемент …

а) ::before;
б) ::after;
в) ::first-line;
г) ::selection.

30. Наслідування стилів в CSS застосовується …

а) до всіх елементів;
б) до всіх елементів, крім властивостей: margin, padding, border;
в) до всіх елементів, крім властивостей: margin, color, border;
г) до всіх елементів, крім властивостей: position, padding, border.

31. Який колір для тексту

* Текст *

зададуть такі стилі:

```
#myId span {color: green;}  /* зелений колір тексту */
div .myClass {color: red;}  /* червоний колір тексту */
div span{color: gray;} /* сірий колір тексту */
```

а) зелений;
б) червоний;
в) сірий;
г) чорний.

32. Скасувати пріоритет стилів можна за допомогою ключового слова …

а) importantly;
б) fatefully;
в) important;
г) big.

33. Розмір шрифта можна задати за допомогою …

а) 7 ключових слів (*small*, *large*, *medium*, …);
б) задання розміру в пікселях або відсотках;
в) задання розміру в одиницях *em*;
г) всіма переліченими способами.

34. Задати для заголовків *h2* горизонтальне зміщення тіні тексту на 5 пікселів вправо, вертикальне зміщення вниз – 4 пікселі, ступінь розмитості – 3 пікселі та колір для тіні #999 можна за допомогою стилю …

а) h2{ text-shadow: 5px 4px 3px #999; };
б) h2{ text-shadow: 4px 5px 3px #999; };
в) h2{ text-shadow: 3px 4px 5px #999; };
г) h2{ text-shadow: 5px 3px 4px #999; }.

35. CSS-властивості *margin* та *padding* елементів можуть бути …

а) тільки додатні значення;
б) і додатні, і від'ємні значення;
в) для властивості *padding* тільки додатні, для *margin* допускаються і від'ємні;
г) для властивості *margin* тільки додатні, для *padding* допускаються і від'ємні.

36. Значення *solid* властивості *border-style* задає границю у вигляді …

а) суцільної лінії;
б) послідовності точок;
в) штрих-лінії;
г) двох паралельних ліній.

37. Значення *double* властивості *border-style* задає границю у вигляді …

а) суцільної лінії;
б) послідовності точок;
в) штрих-лінії;
г) двох паралельних ліній.

38. Значення *hidden* властивості *overflow* дозволяє налаштувати прокрутку блоку наступним чином:

а) якщо контент виходить за межі блоку, то створюється прокрутка; в інших випадках смуги прокрутки не відображаються;
б) відображається тільки видима частина контенту; контент, який виходить за межі блоку, не відображається, а смуги прокрутки не створюються;
в) в блоці відображаються смуги прокрутки, навіть якщо весь контент поміщається в межах блоку, і смуг прокрутки не потрібно;
г) контент відображається, навіть якщо він виходить за межі блоку, смуги прокрутки при цьому не створюються.

39. Значення *visible* властивості *overflow* дозволяє налаштувати прокрутку блоку наступним чином:

а) якщо контент виходить за межі блоку, то створюється прокрутка; в інших випадках смуги прокрутки не відображаються;
б) відображається тільки видима частина контенту; контент, який виходить за межі блоку не відображається, а смуги прокрутки не створюються;
в) в блоці відображаються смуги прокрутки, навіть якщо весь контент поміщається в межах блоку, і смуг прокрутки не потрібно;
г) контент відображається, навіть якщо він виходить за межі блоку, смуги прокрутки при цьому не створюються.

40. Який колір для тексту
* Текст *
зададуть такі стилі:

#myId {color: green;} / зелений колір тексту */*
div .myClass {color: red;} / червоний колір тексту */*
div span{color: gray;} / сірий колір тексту */*

а) зелений;
б) червоний;
в) сірий;
г) чорний.

10.3. Тести по JavaScript

1. Коректні імена змінних у JavaScript:

а) $ds, s_h, s1, $gs_;
б) asd, vasia, with, fgh;
в) r#h, df, as, er;
г) $ds, s_%h, s1, _gs_.

2. Типи даних JavaScript:

а) float, double, int, string, object, char, boolean, null, undefined;
б) float, number, string, object, char, boolean, null, undefined;
в) number, string, object, boolean, null, undefined;
г) float, double, int, string, char, boolean, null.

3. Після виконання коду *var isAlive; console.log(isAlive);***в консоль буде виведено …**

а) не буде нічого;
б) null;
в) undefined;
г) 0.

4. Після виконання коду *x="45"; var z=x + 5;* **змінна z міститиме значення …**

а) 50;
б) 455;
в) undefined;
г) null.

5. Після виконання нижченаведеного коду в консоль буде записано …

var income = 45.8; console.log(typeof income);

а) float;
б) double;
в) number;
г) 45.8.

6. Після виконання нижченаведеного коду в консоль буде записано …

var income = 100;
var strIncome = "100";
var result = income == strIncome;
console.log(result);

а) 100;
б) "100";
в) true;
г) false.

7. Після виконання нижченаведеного коду в консоль буде записано …

var car = {};
if (car) console.log(false); else console.log(true);

а) true;
б) false.

8. Результатом виконання нижченаведеного коду буде …

let z = 20; { let z = 30; console.log(z); } console.log(z);

а) помилка, оскільки змінна Z два рази оголошена;
б) запис в консоль 30 та 20.

9. Результатом виконання нижченаведеного коду буде …

var z = 20; { var z = 30; console.log(z); } console.log(z);

а) запис в консоль 30 та 30;
б) запис в консоль 30 та 20.

10. Після виконання нижченаведеного коду в консоль буде записано …

```
function bar() { foo = "25"; }
bar();
console.log(foo);
```

а) 25;
б) undefined;
в) буде помилка.

11. Після виконання нижченаведеного коду в консоль буде записано …

```
function outer() {  let x = 5;
  function inner() {  x++;  console.log(x); };
  return inner;  }
let fn = outer(); fn();
```

а) 6;
б) undefined;
в) нічого не буде, бо буде помилка.

12. Після виконання нижченаведеного коду в консоль буде записано …

```
function display(){ console.log("0");
display = function(){ console.log("1"); } }
var displayMessage = display;
display();  display();
displayMessage(); displayMessage();
```

а) 1 1 1 1;
б) 0 1 1 1;
в) 0 1 0 0;
г) 1 1 0 0.

13. Після виконання нижченаведеного коду в консоль буде записано …

```
console.log(foo);
var foo = "Tom";
```

а) undefined;
б) Tom;
в) помилка на першому рядку коду *ReferenceError: foo is not defined*.

14. Після виконання нижченаведеного коду в консоль буде записано …

*function change(x) { x = 2 * x; console.log(x); }*
var n = 10; console.log(n); change(n); console.log(n);

а) 10 10 10;
б) 10 20 10;
в) 10 20 20.

15. Після виконання нижченаведеного коду в консоль буде записано …

function change(user){ user.name = "Tom"; }
var bob ={ name: "Bob" }; console.log(bob.name);
change(bob); console.log(bob.name);

а) Bob Tom;
б) Bob Bob.

16. Після виконання нижченаведеного коду в консоль буде записано …

function change(array) { array[0] = 8; }
var numbers = [1, 2, 3]; change(numbers);
console.log(numbers);

а) [8, 2, 3];
б) [1, 2, 3].

17. Для отримання значення року столиці об'єкту країни *var country={ name: "Германия", capital:{ name:"Берлин", year:1237}};* **можна використати рядок коду …**

а) country["capital"]["year"];
б) country.capital.year;
в) country.capital["year"];
г) відповіді "а" та "б" правильні;
д) всі відповіді правильні.

18. У мові програмування JavaScript …

а) конструктор – це звичайна функція;
б) конструктор – це функція, в якій можна встановити властивості та методи з використанням this;
в) немає конструкторів.

19. Властивість *prototype* дає можливість …

а) визначити спільні властивості для об'єктів одного типу;

б) створити статичний об'єкт;

в) створити екземпляр класу.

20. За замовчуванням, усі властивості об'єкту …

а) загальнодоступні;

б) закриті.

21. Після виконання нижченаведеного коду в консоль буде виведено …

function User (name) { this.name = name; var _age = 1; }
var tom = new User("Том"); console.log(tom._age);

а) undefined;

б) 1;

в) помилка буде;

г) null.

22. Чи дозволяє JS принцип успадкування?

а) так;

б) ні.

23. Чи є в JS можливість створення класів?

а) так;

б) ні.

24. Оператор *super* дає можливість

а) викликати функціонал конструктора батьківського класу;

б) позначити клас із суперможливостями;

в) створити помилку, оскільки такого оператора немає.

25. Конструктор класу може мати такий синтаксис:

а) class Person{ constructor(name, age){
this.name = name; this.age = age; };

б) class Person{ Person(name, age){
this.name = name; this.age = age; };

в) немає конструкторів у класах;
г) немає ні класів, ні конструкторів.

26. У регулярних виразах символ /D відповідає …
а) алфавітно-цифровому символу;
б) не алфавітно-цифровому символу;
в) десятковій цифрі;
г) символу, який не є десятковою цифрою.

27. У регулярних виразах символ /W відповідає …
а) алфавітно-цифровому символу;
б) не алфавітно-цифровому символу;
в) десятковій цифрі;
г) символу, який не є десятковою цифрою.

28. Після виконання нижченаведеного коду *nodes* міститиме …
```
var articleDiv = document.querySelector("div.article");
var nodes = articleDiv.childNodes;
```
а) текст, атрибути та самі теги дочірніх елементів елемента *div.article*;
б) дочірні елементи елемента *div.article*;
в) атрибути та самі теги дочірніх елементів *div.article*.

29. Наступний код дозволяє вивести …
```
function getChildren(elem){
   for(var i in elem.childNodes){
      if(elem.childNodes[i].nodeType===1){
         console.log(elem.childNodes[i].tagName);
         getChildren(elem.childNodes[i]);
} } }
var root = document.documentElement;
console.log(root.tagName);  getChildren(root);
```
а) всі елементи html-документа;
б) всі атрибути;
в) вмістиме кожного елемента сторінки;
г) все перераховане.

30. Подія *mousedown* виникає при …

а) знаходженні покажчика миші на елементі, коли кнопка миші знаходиться в натиснутому стані;

б) знаходженні покажчика миші на елементі під час відпускання кнопки миші;

в) входженні покажчика миші в межі елемента;

г) проходженні покажчика миші над елементом.

31. Подія *mouseup* виникає при …

а) знаходженні покажчика миші на елементі, коли кнопка миші знаходиться в натиснутому стані;

б) знаходженні покажчика миші на елементі під час відпускання кнопки миші;

в) входженні покажчика миші в межі елемента;

г) проходженні покажчика миші над елементом.

32. Подія *mouseover* виникає при …

а) знаходженні покажчика миші на елементі, коли кнопка миші знаходиться в натиснутому стані;

б) знаходженні покажчика миші на елементі під час відпускання кнопки миші;

в) входженні покажчика миші в межі елемента;

г) проходженні покажчика миші над елементом.

33. Подія *mousemove* виникає при …

а) знаходженні покажчика миші на елементі, коли кнопка миші знаходиться в натиснутому стані;

б) знаходженні покажчика миші на елементі під час відпускання кнопки миші;

в) входженні покажчика миші в межі елемента;

г) проходженні покажчика миші над елементом.

34. Правильний запис json-об’єкта:

а) { name: "Tom", married: true, age: 30 };

б) { "name": "Tom", "married": true, "age": 30 };

в) { name: "Tom", married: true, "my age": 30 };

г) всі перераховані.

35. За замовчуванням, час життя cookie триває до …

а) 24 годин;

б) до закриття браузера;

в) до зачищення вручну куків;

г) немає часу, за замовчуванням, обов'язково треба визначати.

36. Яка з типів браузерної пам'яті передається по http?

а) session storage;

б) local storage;

в) cookie;

г) всі перераховані.

37. Session storage …

а) реалізує тимчасове зберігання інформації, яка видаляється після закриття браузера;

б) являє собою сховище даних на постійній основі;

в) нічого не означає, оскільки немає такого типу пам'яті.

38. Local storage …

а) реалізує тимчасове зберігання інформації, яка видаляється після закриття браузера;

б) являє собою сховище даних на постійній основі;

в) нічого не означає, оскільки немає такого типу пам'яті.

10.4. Тести по PHP

1. Найпоширенішими веб-сервером та СКБД, які працюють у зв'язці з мовою PHP, є …

а) IIS, MsSQL Server;

б) Nginx, PostgreSQL;

в) Apache, MySQL;

г) Nginx, MsSQL Server.

2. Відправлені дані з HTML-форми на мові PHP можна отримати з глобальних масивів …

а) $_GET, $_POST;
б) $_GET, $_ECHO;
в) $_POST, $_ECHO;
г) $_ECHO, $_FORM.

3. Оператором конкатенації (об'єднання) рядків на мові PHP є …

а) ++;
б) +;
в) .;
г) concat().

4. Змінними на PHP є …

а) $_for, $&gh, $gh;
б) $_4for, $h_dsg3, $while;
в) $4for, $h_dsg3, $ghf;
г) _gh4, $for, $f3g4.

5. Нижченаведений php-код виведе на веб-сторінку …

*$i=4; echo "квадрат числа $i дорівнює ". $i * $i;*

а) квадрат числа 4 дорівнює 16;
б) квадрат числа 4 дорівнює 4*4;
в) квадрат числа $i дорівнює 16;
г) квадрат числа $i дорівнює 4*4.

6. Мова PHP підтримує такі типи даних:

а) boolean, integer, float, string, array, object, resource, NULL;
б) boolean, integer, double, float, string, array, object, NULL;
в) boolean, int, double, string, array, object, resource, NULL;
г) boolean, integer, double, string, array, object, resource, NULL.

7. Для передачі змінної *a* за посиланням потрібно записати …

а) function get(ref $a){ тіло функції };

б) function get(&$a){ тіло функції };
в) function get(a){ тіло функції };
г) function get(out $a){ тіло функції }.

8. Для доступу до змінної, яка визначена у деякій функції, з будь-якого місця php-коду, її потрібно визначити з ключовим словом …

а) static;
б) public;
в) global;
г) var.

9. Для підключення php-файлу до іншого php-файлу використовують …

а) include;
б) require;
в) include_once;
г) всі перелічені.

10. Для підключення php-файлу до іншого php-файлу з виведенням помилки у разі відсутності файлу та забезпеченням тільки разового підключення використовується …

а) include;
б) require_once;
в) include_once;
г) всі перелічені;

11. Для виводу розміру масиву *$a* в php використовується …

а) count($a), sizeof($a);
б) length($a), sizeof($a);
в) count($a), length($a);
г) count($a), length($a), sizeof($a).

12. Для сортування елементів масиву рядків по алфавіту або масиву чисел за зростанням в php використовується …

а) asort();

б) arsort();
в) ksort();
г) krsort().

13. Для сортування елементів масиву чисел за спаданням в php використовується …

а) asort();
б) arsort();
в) ksort();
г) krsort().

14. Для сортування елементів масиву за ключами в php використовується …

а) asort();
б) arsort();
в) ksort();
г) kksort().

15. Для отримання cookie в php можна використовувати глобальний асоціативний масив …

а) _COOKIE;
б) COOKIE;
в) $_COOKIE_START;
г) $_COOKIE.

16. Коректне визначення cookie:

а) setcookie("language", "китайський", time() + 3600);
б) setcookie("language", "китайський");
в) setcookie("language", "китайський", time() + 3600, "/");
г) всі перелічені.

17. Вираз *setcookie("language","китайський", time() + 3600)* **визначає cookie на …**

а) годину;
б) хвилину;
в) 36 секунд;
г) 36 хвилин.

18. Для запуску сесії потрібно викликати функцію …

а) session_enter();
б) session_start();
в) session_push();
г) session_begin().

19. Для запису або читання змінної сесії використовується глобальний асоціативний масив …

а) _SESSION;
б) $_SESSION;
в) $SESSION;
г) $_SESSION_START.

20. Параметри-значення набору в запиті get відокремлюються знаком …

а) &;
б) $;
в) +;
г) ;.

21. Результатом виконання коду *echo 11 . 22;* **буде вивід на веб-сторінці …**

а) 33;
б) 1122;
в) 11.22;
г) 11.

22. Для знищення змінної *$a* використовується оператор …

а) unset($a);
б) isset($a);
в) delete($a);
г) drop($a).

23. Для перевірки ініціалізованості змінної $a застосовується оператор …

а) unset($a);
б) isset($a);

в) isvariable($a);
г) isvar($a).

24. Результатом виконання коду *$a=NULL; echo $a;* **на веб-сторінці буде …**

а) вивідення 0;
б) відсутність якогось виведення;
в) виведення NULL;
г) помилка.

25. Результатом виконання коду *$a=4;echo -$a--;* **буде вивід на веб-сторінці числа …**

а) 4;
б) 3;
в) -3;
г) -4.

26. Результатом виконання нижченаведеного коду буде вивід на веб-сторінці …

```
$a = 10; $b = 5;
echo "$a + $b " . '$a + $b';
```

а) 10 + 5 10 + 5;
б) $a + $b $a + $b;
в) 15 $a + $b;
г) 10 + 5 $a + $b.

27. Для визначення деякої константи NUMBER потрібно записати …

а) var NUMBER=22;
б) final NUMBER=22;
в) const NUMBER=22;
г) define("NUMBER", 22).

28. Результатом виконання нижченаведеного коду буде вивід на веб-сторінці …

```
$a = 10=="10";
$b = 10!=="10";
echo $a . $b;
```

а) порожніх символів;
б) 11;
в) 10;
г) 01.

29. Результатом виконання нижченаведеного коду буде вивід на веб-сторінці …

```
echo 7 > 6 && 8 > 8 or 9 > 8;
echo 1 || 0 xor 0;
```

а) порожніх символів;
б) 11;
в) помилки;
г) 1.

30. Результатом виконання нижченаведеного коду буде вивід на веб-сторінці …

```
$n=10;
for($i=0;$i<$n;$i++,$n--);
    echo $i;
```

а) 5;
б) 0;
в) 0123456789;
г) 01234.

СПИСОК ЛІТЕРАТУРИ

1. Грицюк Ю.І. Аналіз вимог до програмного забезпечення. Львів : Вид-во НУ “Львівська політехніка”, 2018. 456 с.

2. Мартін Р. Чиста архітектура. Харків : Фабула, 2019. 368 с.

3. Мартін Р. Чистий код. Харків : Фабула, 2019. 416 с.

4. Мельник Р.А. Програмування веб-застосувань (фронт-енд та бек-енд). Львів : Вид-во НУ “Львівська політехніка”, 2018. 248 с.

5. Посібник з HTML5 і CSS3. URL: https://metanit.com/web/html5/.

6. Український веб-довідник. URL: https://css.in.ua/.

7. Bootstrap. Documentation. URL: https://getbootstrap.com/docs/4.5/getting-started/introduction/.

8. Посібник з JavaScript. URL: https://metanit.com/web/javascript/.

9. Мова програмування JavaScript. URL: https://abitap.com/category/javascript/.

10. Онлайн-книга “Вивчаємо jQuery”. URL: https://metanit.com/web/jquery/.

11. Посібник з PHP. URL: https://metanit.com/web/php/.

Навчальне видання

Двірничук *Костянтин Васильович*
Вацек *Діана Орестівна*

ВЕБ-ПРОГРАМУВАННЯ ТА ВЕБ-ДИЗАЙН

Навчальний посібник

Відповідальний за випуск ***Г.І Воробець***

Літературний редактор ***О.В. Колодій***

Комп'ютерний набір ***К.В. Двірничук, Д.О. Вацек***

Технічний редактор ***О.М. Кудрінська***

Підписано до друку 26.12.2022. Формат 60x84/16
Папір офсетний. Друк різографічний. Ум.-друк. арк. 25,8.
Обл.-вид. арк. 27,8. Зам. Н-133.
Видавництво та друкарня Чернівецького національного університету
імені Юрія Федьковича
58002, м.Чернівці, вул. Коцюбинського, 2
e-mail: ruta@chnu.edu.ua

Свідоцтво суб'єкта видавничої справи ДК №891 від 08.04.2002 р.