

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДВНЗ «УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ
ТА ТЕХНОЛОГІЙ

Пецко В. І.

ОСНОВИ ВЕБРОЗРОБКИ ТА ВЕБПРОГРАМУВАННЯ

Електронний навчальний посібник

УЖГОРОД – 2026

УДК 004.774: 004.43

П 30

Пецко В.І. Основи веброзробки та вебпрограмування: електронний навчальний посібник для студентів спеціальності F3 «Комп'ютерні науки». – Ужгород: ДВНЗ «УжНУ», 2026. 175 с. [Електронне видання]. – ISBN 978-617-8576-21-9

У електронному навчальному посібнику до курсу «Основи веброзробки та вебпрограмування» наведено матеріал з сучасних та перспективних інтернет-технологій, викладені основні підходи, принципи, методи та засоби верстки сайтів за допомогою мови гіпертекстової розмітки HTML5, каскадних таблиць стилів CSS3, система контролю версій Git та основи JS.

Автор:

Пецко В.І., к.т.н., доцент, доцент кафедри інформаційних управляючих систем та технологій ДВНЗ «Ужгородський національний університет»

Рецензенти:

Гече Ф.Е., д.т.н., професор, професор ДВНЗ «Ужгородський національний університет»;

Головач Й.І., д.т.н., професор, професор Закарпатського угорського інституту імені Ференца Ракоці II.

*Рекомендовано до друку та опублікування
Вченою радою ДВНЗ «УжНУ»
(протокол №3 від 5 березня 2026 р.)*

ISBN 978-617-8576-21-9

© В.І. Пецко, 2026

vasyl.petsko@uzhnu.edu.ua

ЗМІСТ

ВСТУП.....	5
1. HTML.....	6
1.1. Вебсторінка та вебсайт	6
1.2. Структура HTML-документа.....	8
1.3. Задання кольору в HTML-документі	13
1.4. Елементи для форматування тексту	14
1.5. Блочні елементи.....	16
1.6. Списки	18
1.7. Текстові гіперпосилання	21
1.8. Таблиці	26
1.9. Зображення	31
1.10. Аудіо та відео на вебсторінках.....	34
1.11. Форми в HTML	38
2. КАСКАДНІ ТАБЛИЦІ СТИЛІВ	47
2.1. Основи CSS.....	47
2.2. Синтаксис CSS	52
2.3. Селектори	59
3. СИСТЕМА КОНТРОЛЮ ВЕРСІЙ GIT	63
3.1 Установка Git.....	63
3.2 Налаштування SSH-ключів	65
3.3 Реєстрація на GitHub та настройка аккаунта та репозиторії.....	66
3.4 Створення репозиторію	67
3.5 Git: ігнорування файлів.....	68
3.6 Додавання та видалення файлів в індексі Git.....	70
3.7 Робота з файлами в Git: додавання, індексація та коміти	71
3.8 Робота з командою git log та перегляд історії комітів.....	72
3.9 Робота з гілками в Git.....	74
3.10 Основні команди Git:	75
4. JAVASCRIPT	78
4.1. Способи додавання скриптів на сторінку.....	79
4.2. Опис змінних та констант	82

4.3. Типи даних.....	85
4.4. Логічні оператори.....	95
4.5. Ввід та вивід даних. Взаємодія з користувачем.....	97
4.6. Умовні оператори.....	99
4.7. Цикли.....	102
4.8. Функції.....	106
4.9. Масиви	109
5. БРАУЗЕРНЕ СЕРЕДОВИЩЕ	116
5.1. Структура браузерних об'єктів	116
5.2. Дерево DOM	118
5.3. Навігація по DOM-елементах	120
5.4. Основи роботи з подіями	140
6. ЗАВДАННЯ ДЛЯ ЛАБОРАТОРНИХ РОБІТ	151
6.1. Лабораторна робота №1	151
6.2. Лабораторна робота №2	155
6.3. Лабораторна робота №3	163
6.4. Лабораторна робота №4	164
6.5. Лабораторна робота №5	168
ПЕРЕЛІК ПИТАНЬ ДЛЯ САМОКОНТРОЛЮ.....	172
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	174

ВСТУП

На сьогодні web-технології займають важливе місце у сучасному інформаційному суспільстві. За їх допомогою переважна більшість людей дізнається про важливі події у світі та країні, звичною стала купівля речей у онлайн магазинах, широкого розповсюдження набув Internet-речей, активно використовуються системи онлайн навчання (Moodle) та освітні платформи такі як Coursera та Edx.com тощо.

Основними завданнями вивчення дисципліни «Основи веброзробки та вебпрограмування» є формування у студентів вмінь та навичок створювати сайти з використанням набутих знань HTML5, CSS3, GIT та JavaScript.

Метою представленого навчально посібника є ознайомлення з основними підходами, принципами, методами та засобами верстки сайтів за допомогою мови гіпертекстової розмітки HTML5, каскадних таблиць стилів CSS3, системи контролю версій GIT та основ JavaScript.

У посібнику пропонуються зрозумілі пояснення, кожен розділ супроводжується численними прикладами вебсторінок і фрагментами коду, а також завдання для формування навичок, що необхідні для створення власних вебсайтів та вебдодатків. Цей посібник не лише надає базові знання з HTML, CSS і JavaScript, а й розглядає їх взаємодію.

1. HTML

1.1. Вебсторінка та вебсайт

Гіпертекст — це така форма організації тексту, при якій його одиниці представлені не в лінійній послідовності, а як система явно вказаних можливих переходів, зв'язків між ними. Слідуючи цим зв'язкам, можна читати матеріал в будь-якому порядку, утворюючи різні лінійні тексти.

Гіпертекст – це текст, у який вбудовані спеціальні коди, що задають форматування тексту, наявність у ньому ілюстрацій, мультимедійних вставок та гіперпосилань. Ці коди визначені у мові програмування **HTML (Hyper Text Markup Language** – мова гіпертекстової розмітки).

Гіпертекстові документи, що розміщуються у всесвітній павутині, називаються **вебсторінками**. Сукупність вебсторінок, що тематично пов'язані між собою й розроблені як єдине ціле, називають **вебсайтом** або просто **сайтом**.

Для завантаження та перегляду вебсторінок використовуються спеціальні програми – **браузери** (IE, Firefox, Chrome, Opera, Safari та ін). Своєю появою Веб має завдячити **Тіму Бернесу-Лі**, який винайшов протокол **HTTP**, адреси **URL** та мову **HTML**.

Гіпертекстове посилання (гіперпосилання, гіперзв'язок чи гіперлінк) – це об'єкт вебсторінки, що містить інформацію про адресу іншої вебсторінки або про певне місце на поточній. Таким об'єктом може бути фрагмент тексту (зазвичай виділений

кольором та підкресленням) або ілюстрація.

Процес переходу в інші місця поточної вебсторінки або до інших вебсторінок за допомогою гіперпосилань називають **навігацією**.

Методи розробки сайтів:

- **методи ручного написання сайтів** на одній або декількох мовах веб-програмування, зокрема HTML та CSS, з можливим включенням Javascript, Flash та ін. Для створення серверних скриптів, застосовують спеціальні мови PHP, ASP.NET і т.д.;
- **методи автоматизованого створення сайтів:** за допомогою спеціальних конструкторів сайтів або ж систем керування контентом (**CMS**), що являє собою певну готову візуальну й програмну оболонку, яку користувач може заповнити необхідним контентом, а також за своїм бажанням змінити й налаштувати.

1.2. Структура HTML-документа

HTML-документ складається з основного тексту і тегів розмітки. Фактично це звичайний текстовий файл, для створення і редагування якого можна використати будь-який текстовий редактор. Зазвичай HTML-документи містяться у файлах із розширенням .html.

Елемент **<!DOCTYPE>** призначений для вказівки типу поточного документа - **DTD** (document type definition, опис типу документа). Це необхідно, щоб браузер розумів, як слід інтерпретувати поточну вебсторінку, адже HTML існує в декількох версіях, крім того, є **XHTML** (EXtensible HyperText Markup Language, розширена мова розмітки гіпертексту), схожа на HTML, але відрізняється від неї по синтаксису. Щоб браузер не плутався і розумів, згідно якого стандарту відображати сторінку, необхідно в першому рядку коду задавати **<!DOCTYPE>**.

Синтаксис:

```
<!DOCTYPE [елемент верхнього рівня] [Публічність]
"[Реєстрація]//[Організація]//[Тип] [Ім'я]//[Мова]" "[URL]">
```

Параметри:

Елементи верхнього рівня — показують елемент верхнього рівня у документі, для HTML це тег <html>.

Публічність — об'єкт є публічним (значення PUBLIC) або системним ресурсом (значення SYSTEM), наприклад, таким як локальний файл. Для HTML/XHTML вказується значення PUBLIC.

Реєстрація — повідомляє, що розробник DTD зареєстрований у міжнародній організації по стандартизації (International Organization for Standardization, ISO). Приймає одне з двох значень: плюс (+) - розробник зареєстрований в ISO і - (мінус) - розробник не зареєстрований. Для W3C значення ставиться «-».

Організація — унікальна назва організації, яка розробила DTD. Офіційно HTML/XHTML публікує W3C, ця назва і пишеться в `<!DOCTYPE>`.

Таблиця 1.1. – Основні види `<!DOCTYPE>` html

HTML 4	
<code><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd"></code>	Строгий синтаксис HTML
<code><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"></code>	Перехідний синтаксис HTML
<code><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd"></code>	У документі застосовуються фрейми
HTML 5	
<code><!DOCTYPE html></code>	У цій версії тільки один тип
XHTML	

<pre><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd" ></pre>	Вважається, що XHTML витіснить з часом HTML. Ніякого поділу на типи немає
---	---

Тип – тип описуваного документа.

Для HTML/XHTML значення вказується DTD.

Ім'я – унікальне ім'я документа для опису DTD.

Мова – мова, якою написаний текст для опису об'єкта. Містить дві літери, пишеться у верхньому регістрі. Для документа HTML/XHTML вказується англійська мова (EN).

URL – адреса документа з DTD.

Різниця між строгим і перехідним описом документа полягає в різному підході до написання коду документа. Строгий HTML вимагає жорсткого дотримання специфікації HTML і не дозволяє помилок. Перехідний HTML спокійно ставиться до деяких погрішностей коду.

Наприклад, в строгому HTML і XHTML необхідна наявність тега **<title>**, а в перехідному HTML його можна опустити і не вказувати. При цьому пам'ятаємо, що браузер в кожному разі покаже документ, незалежно від того, відповідає він синтаксису чи ні, але за наявності помилок результат буде непередбачуваний.

Основну структуру HTML-документа визначають чотири парні теги: **<html>**, **<head>**, **<title>**, **<meta>**. Їхня наявність передбачена у всіх таких HTML-документах.

Структура HTML-документа.

```
<!DOCTYPE      HTML      PUBLIC      "-//W3C//DTD      HTML      4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
  < head >
    < title > Заголовок документа </ title >
  </ head >
  < body >
    Текст, що відображається на екрані
  </ body >
</html>
```

Елемент **<html>** є контейнером, який містить в собі весь вміст сторінки. Як правило, **<html>** йде в документі другим, після **<!DOCTYPE>**. Закриваючий тег **</html>** повинен завжди стояти в документі останнім. Все, що йде після нього – ігнорується.

Елемент **<head>** призначений для зберігання інших елементів, мета яких - допомогти браузеру в роботі з даними. Також всередині контейнера **<head>** знаходяться метатеги, які використовуються для зберігання інформації призначеної для браузерів і пошукових систем.

Вміст **<head>** не відображається безпосередньо на вебсторінці, за винятком **<title>**. Всередині контейнера **<head>** допускається розміщувати такі елементи: **<base>**, **<basefont>**, **<bgsound>**, **<link>**, **<meta>**, **<script>**, **<style>**, **<title>**.

Елемент **<title>** визначає заголовок документа. Він не є частиною документа і не показується прямо на сторінці. Допускається використовувати тільки один тег **<title>** на документ і розміщувати його тільки в контейнері **<head>**.

Елемент **<meta>** визначає метатеги для зберігання інформації призначеної для браузерів і пошукових систем. Наприклад, механізми пошукових систем звертаються до метатегів для отримання опису сайту, ключових слів та інших даних. Дозволяється використовувати більше, ніж один метатег, всі вони розміщуються в контейнері **<head>**. Елемент не має закриваючого тега.

Елемент **<body>** призначений для зберігання змісту сторінки (контенту), що відображається у вікні браузера. Всю інформацію в документі слід розташовувати саме всередині контейнера **<body>**. До такої інформації відноситься: текст, зображення, теги, скрипти Javascript і т.д.

Основні атрибути елемента body:

text – задає колір тексту;

bgcolor – колір фону;

link – колір посилання;

alink – колір активного посилання;

vlink – колір відвіданого посилання.

1.3. Задання кольору в HTML-документі

В HTML **колір** задається одним з двох шляхів: за допомогою шістнадцяткового коду чи за назвою для деяких основних кольорів. Переважно використовується спосіб, заснований на шістнадцятковій системі числення, як більш універсальний.

Щоб не виникало плутанини у визначенні системи числення, перед шістнадцятковим числом ставиться символ решітки #, наприклад #fa8e47. Наступний опис задає помаранчевий колір тла сторінки: `<body bgcolor="#fa8e47">`. Перші дві цифри (FA) визначають червону складову кольору, цифри з третьої по четверту (8E) – зелену, а останні дві цифри (47) – синю. Кожен з трьох кольорів - червоний, зелений і синій - може приймати значення від 00 до FF, що в результаті утворює 256 відтінків. Таким чином, загальна кількість кольорів може бути $256 \times 256 \times 256 = 16.777.216$ комбінацій. Колірна модель, заснована на червоній, зеленій і синій складових отримала назву **RGB** (red, green, blue; червоний, зелений, синій). Ця модель адитивна, при якій сумування всіх трьох компонент утворює білий колір.

background – задає малюнок, який буде використано для фону;
bgproperties="fixed" – забороняє прокрутку фонового зображення разом із сторінкою.

Приклад найпростішої сторінки:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
```

```
<meta charset="utf-8">
<title> Моя перша сторінка</title>
</head>
<body>
    Текст сторінки
</body>
</html>
```

1.4. Елементи для форматування тексту

Будь-яка кількість пробілів, що йдуть підряд, в браузері відображаються як один. Немає переносів у тексті. HTML не підтримує розстановку переносів у словах, як це роблять текстові редактори, всі слова пишуться без розбиття. Короткі рядки при цьому розтягуються за рахунок автоматичного додавання пробілів між словами. Текст займає ширину вікна браузера. Якщо написати один довгий рядок в коді HTML, то в браузері він буде відформатований, щоб текст помістився по ширині в вікно. Якщо введений рядок без пробілів і він ширше вікна браузера, то з'явиться горизонтальна смуга прокручування.

Для форматування виводу тексту застосовуються наступні елементи:

**** - написання тексту жирним шрифтом.

Приклад: Текст. **Жирний текст.** Текст.

Виведеться наступним чином: Текст. **Жирний текст.** Текст.

**** - призначений для акцентування тексту. Браузери відображають такий текст жирним накресленням.

<i> - вивід тексту курсивом.

**** - призначений для акцентування тексту. Браузери відображають такий текст курсивом.

Слід зазначити, що теги **<i>** і ****, також як **** і ****, незважаючи на схожість результату, є не зовсім еквівалентними і замінюваними. Перший тег **<i>** - є тегом фізичної розмітки і встановлює курсивний текст, а тег **** - тегом логічної розмітки і визначає важливість поміченого тексту. Такий поділ тегів на логічне і фізичне форматування призначався, щоб зробити HTML універсальним, в тому числі незалежним від пристрою виводу інформації. Теоретично, якщо скористатися, наприклад, мовним браузером, то текст, оформлений за допомогою тегів **<i>** і ****, буде відзначений по-різному. Однак вийшло так, що в популярних браузерах результат використання цих тегів рівнозначний.

<sub> - нижній індекс. Текст розташовується нижче базової лінії інших символів рядка і зменшеного розміру - H₂O.

<sup> - верхній індекс - $x^2+bx+c=0$.

<big> збільшує розмір шрифту на одиницю в порівнянні зі звичайним текстом. В HTML розмір шрифту вимірюється в умовних одиницях від 1 до 7, середній розмір тексту, що використовується за замовчуванням, прийнятий 3. Таким чином, додавання тега **<big>** збільшує текст на одну умовну одиницю.

<small> - зменшує розмір шрифту на одиницю. Протилежний до **<big>**.

**
** - перехід на новий рядок в тому місці, де цей тег розташований. На відміну від тега абзацу **<p>**, використання тега

 не додає порожній відступ перед рядком. Якщо текст, в якому використовується перехід рядка, обтікає плаваючий елемент, то за допомогою атрибуту **clear** тега
 можна зробити так, щоб наступний рядок починався нижче елемента. Атрибут clear повідомляє браузеру, як обробляти наступний рядок, якщо текст обтікає плаваючий елемент.

1.5. Блочні елементи

Елемент <div> відноситься до універсальних блочних контейнерів і застосовується в тих випадках, де потрібні блочні елементи без додаткових властивостей.

Елемент <p> визначає параграф (абзац) тексту. Завжди починається з нового рядка. Абзаци тексту, що йдуть один за одним розділяються між собою порожнім рядком. Якщо закриваючого тега немає, вважається, що кінець абзацу збігається з початком наступного блокового елемента. Якщо відступи між абзацами не потрібні, то для переходу на новий рядок слід використовувати
.

<h1>, ..., <h6> - ця група тегів визначає текстові заголовки різного рівня. Заголовки виконують важливу функцію на вебсторінці. По-перше, вони показують важливість розділу до якого відносяться. Чим більше заголовків і його рівень, тим більше він значимий. По-друге, за допомогою різних заголовків легко регулювати розмір тексту. Чим вище рівень заголовка, тим більше розмір шрифту. Верхнім рівнем є рівень 1 (<h1>), а нижнім – 6. І,

по-третє, пошукові сервіси додають рейтинг тексту, якщо він знаходиться всередині тега заголовка. Це важливо для потрапляння в перші рядки результату в пошуковій системі за ключовими словами.

За допомогою атрибуту **align**, у будь-якому з наведених вище блоків, можна вирівнювати текст всередині цього контейнера. Він може приймати одне з наступних значень **left** (по лівому краю), **center** (по центру), **right** (по правому краю), **justify** (по всій ширині контейнера). За замовчуванням встановлено **left**. Також, можливе застосування атрибуту **style**, який записується згідно правил синтаксису CSS і буде розглянуто далі.

Елемент **<pre>** задає блок попередньо форматowanego тексту. Такий текст відображається зазвичай моноширинним шрифтом і з усіма пробілами між словами. В HTML будь-яку кількість пробілів, що йдуть в коді поспіль на вебсторінці відображається як один. Тег **<pre>** дозволяє обійти цю особливість і відображати текст як потрібно розробнику. Наступні теги не повинні розміщуватися всередині контейнера **<pre>**: **<big>**, ****, **<small>**, **<sub>** і **<sup>**.

Елемент **<hr>** малює горизонтальну лінію, яка за своїм виглядом залежить від використовуваних атрибутів. Лінія завжди починається з нового рядка, а після неї всі елементи відображаються у наступному рядку. Тег є одинарним.

Приклад сторінки з форматуванням:

```
Pr_3
<!DOCTYPE html>
```

```

<html>
<head>
    <meta charset="utf-8">
    <title>Форматування тексту</title>
</head>
<body bgcolor="yellow" text="blue">
<h1> Квадратні рівняння </h1>
<h2> Теорема Вієта</h2>
<p><b>Теорема</b> - нехай <i>x<sub>1</sub></i>,
<i>x<sub>2</sub></i> - розв'язки рівняння
<i>ax<sup>2</sup>+bx+c=0</i> тоді. <br>
    <i>x<sub>1</sub>+x<sub>2</sub>=-b/a</i></p>
    ...
<hr>
<div align="center"> Доведення </div>
<p> Нехай &hellip; </p>
<p align="right"> <big>д</big>Доведено &copy; </p>
</body>
</html>

```

1.6. Списки

Для подання тексту в структурованому вигляді, коли кожен абзац виділено за допомогою позначки або номера, призначені списки.

У HTML-документах використовують три види списків:

- неупорядкований (маркерований);
- упорядкований (нумерований);
- список визначень.

У **неупорядкованому списку** для виділення елементів використовуються певні позначки (крапки, квадрати тощо).

Починається такий список із тегу , а закінчується – . Кожний елемент списку починається з тегу . Тег може мати атрибут TYPE, який визначає форму позначки. Цей атрибут набуває таких значень:

disk – зафарбоване коло;

circle – коло;

square – маленький чорний квадрат.

Приклад створення маркерованого списку:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
<title>Маркерований список</title>
</head>
<body>
<hr>
<ul type="square">
<li> пункт 1</li>
<li> пункт 2</li>
<li> пункт 3</li>
<li> пункт 4</li>
</ul>
<hr>
</body>
</html>
```

У **впорядкованому списку** всі елементи пронумеровані. Починається список із тегу , а закінчується – . Кожний елемент такого списку також починається з тегу . Тег може мати атрибут TYPE, який визначає тип нумерації. Цей

атрибут набуває таких значень:

A – велика літера;

a – мала літера;

I – велика римська цифра;

i – мала римська цифра;

1 – арабська цифра.

У разі потреби за допомогою атрибута **START** можна задати відмінний від одиниці початковий номер елемента, наприклад `<OL TYPE="I" START="5">`. Нумерований список, початковий елемент якого позначено латинською літерою E, яка є п'ятою в алфавіті, описують так:

```
<ol TYPE="A" START="5">
<li>Перший елемент списку</li>
<li>Другий елемент списку</li>
<li>Третій елемент списку</li>
<li>Четвертий елемент списку</li>
</ol>
```

Список визначень (definition list). Кожен елемент такого списку складається з двох частин: терміна та його визначення. Сам список задається за допомогою контейнера `<dl>`, термін - тегом `<dt>`, а його визначення - за допомогою тега `<dd>`. Вкладення тегів для створення списку визначень продемонстровано в прикладі.

Приклад загальної структури списку визначень:

```
<dl>
  <dt> Термін 1 </ dt>
    <dd> Визначення 1 </ dd>
  <dt> Термін 2 </ dt>
    <dd> Визначення 2 </ dd>
```

</ dl>

1.7. Текстові гіперпосилання

Гіперпосилання – це вказівка для браузера, до якого об’єкта у межах або поза межами HTML-документа він має звернутися. За допомогою гіперпосилань користувачі можуть переходити з однієї вебсторінки на іншу, завантажувати файли тощо.

**** текст посилання ****

Для створення гіперпосилання необхідно використовувати теги <A> та , визначивши для тегу <A> атрибут HREF. Його значенням має бути адреса URL, на яку вказує посилання. Текст посилання розташовують між тегами <A> і .

Якщо вебсторінка, на яку вказує посилання, розміщена на іншому вебсайті, то значенням атрибута HREF має бути повна URL-адреса з назвою протоколу включно; такі посилання називають **зовнішніми**. Якщо ж гіперпосилання вказує на іншу сторінку того самого сайту, то для пошуку документа достатньо задати лише відносний шлях; таке посилання називають **внутрішнім**.

Невідвідане гіперапосилання. Такий стан характеризується для посилань, які ще не відкривали. За замовчуванням невідвідані текстові посилання зображуються **синього кольору і з підкресленням**.

Активне гіперапосилання. Посилання позначається як активне в момент його відкриття. Активним посилання стає також,

при його виділенні за допомогою клавіатури. Колір такого посилання за замовчуванням **червоний**.

Відвідане гіперпосилання. Як тільки користувач відкриває документ, на який вказує посилання, воно позначається як відвідане і змінює свій колір на **фіолетовий**, встановлений за замовчуванням.

За замовчуванням, при переході за посиланням, документ відкривається в поточному вікні або фреймі. При необхідності, ця умова може бути змінена атрибутом **target** тега `<a>`. Синтаксис наступний:

`<a target="ім'я вікна"> ... </a>`

Як значення використовується ім'я вікна або фрейму, задане атрибутом `name`. Якщо встановлено неіснуюче ім'я, то буде відкрито нове вікно. В якості зарезервованих імен застосовуються такі:

_blank - завантажує сторінку в нове вікно браузера;

_self - завантажує сторінку в поточне вікно (це значення задається за замовчуванням);

_parent - завантажує сторінку у фрейм-батько, якщо фреймів немає, то це значення працює як **_self**;

_top - скасовує всі фрейми і завантажує сторінку в новому вікні браузера, якщо фреймів немає, то це значення працює як **_self**.

У прикладі показано, як зробити, щоб посилання відкривалося в новому вікні:

```
<P><a href="new.html" target="_blank"> Відкрити в новому вікні  
</a> </ p>
```

Атрибут `target` коректно використовувати тільки при перехідному `<!DOCTYPE>`, при строгому `<!DOCTYPE>` буде повідомлення про помилку, оскільки в цій версії HTML `target` вже не підтримується.

Атрибут **`title`** додає пояснювальний текст до посилання у вигляді підказки. Така підказка відображається, коли курсор миші затримується на посиланні, після чого підказка через деякий час пропадає. Синтаксис наступний:

```
<a title="текст"> ... </a>
```

Приклад створення підказки:

```
<P> <a href="zoo.html" title="Малюнки різних тварин і не  
тільки..."> Малюнки </a> </ p>
```

Якщо в атрибуті `HREF` задати адресу електронної пошти зі словом **`mailto:`** перед нею, то після вибору такого посилання можна надіслати електронний лист, де в полі Кому буде записано цю адресу.

```
<p> <a href="mailto:vlad@htmlbook.ru"> Задавайте питання по  
електронній адресі </a> </ p>
```

Якорем називається закладка з унікальним ім'ям на певному місці вебсторінки, призначена для створення переходу до неї за посиланням. Якір зручно застосовувати в документах великого об'єму, щоб можна було швидко переходити до потрібного розділу.

Для визначення якоря використовують теги `<A>` і `</A>`, але замість атрибута `HREF` задають атрибут **`NAME`**, значенням якого

має бути ім'я якоря. Воно може складатися з літер та цифр, але не повинно містити символів пробілу. Якщо на сторінці є кілька міток, то всі вони повинні мати різні назви. Синтаксис наступний:

** **

Для створення посилання на встановлений якорь потрібно у тегу <A> зазначити його ім'я в кінці адреси URL після імені документа, відокремивши його символом #. Символ # вказує на те, що після нього записано назву мітки, а не ім'я файлу.

Посилання на мітку всередині поточного документа задають так:

Текст посилання.

<P> Наверх </ p>

Посилання на мітку всередині іншого документа задають так:

Перейти до нижньої частини тексту

Універсальні атрибути:

- **accesskey** дозволяє отримати доступ до елементу за допомогою заданого сполучення клавіш:
...;
- **class** визначає ім'я класу, яке дозволяє зв'язати тег зі стильовим оформленням: **class="ім'я";**
- **contextmenu** встановлює контекстне меню для елемента:
contextmenu="ідентифікатор";
- **hidden** приховує вміст елемента від перегляду: **<a hidden>;**
- **id** вказує ім'я стильового ідентифікатора: **id="ім'я";**

- **style** застосовується для визначення стилю елемента за допомогою правил CSS: `style="правила описання стилів"`;
- **tabindex** встановлює порядок отримання фокусу при переході між елементами за допомогою клавіші Tab: `tabindex="число"`;
- **title** описує вміст елемента у вигляді підказки: `title="текст"`.

1.8. Таблиці

Таблиця складається з рядків і стовпців, які можуть містити текст і малюнки. Зазвичай таблиці використовуються для впорядкування і представлення даних. За допомогою таблиць зручно верстати макети сторінок, розташувавши потрібним чином фрагменти тексту і зображень.

Створення таблиці.

Для створення таблиці використовують чотири елементи. Таблицю описують за допомогою тегів `<TABLE>...</TABLE>`, вона повинна мати один або кілька рядків `<TR>...</TR>`, у кожному з яких може міститися заголовок `<TH>...</TH>` або дані (комірки з даними) `<TD>...</TD>`. Текст в комірці, оформленої за допомогою тега `<TH>`, відображається браузером шрифтом жирного накреслення і вирівнюється по центру. В іншому, різниці між тегами `<TD>` і `<TH>` немає.

По замовчуванню таблиця має невидимі межі комірок. Для того щоб зробити їх видимими, використовують атрибут **BORDER**, який має цілочислове значення, що визначає товщину рамки в пікселях, наприклад `<TABLE BORDER=5>`.

Таблиця може мати заголовок, який задається тегами `<CAPTION>` та `</CAPTION>`. Тег `<CAPTION>` може містити атрибут **ALIGN** з одним із значень **top** або **bottom**, які визначають розташування заголовка відповідно перед таблицею або після неї (по замовчуванню встановлено значення **top**). Наприклад, `<CAPTION ALIGN=bottom>Розклад уроків</CAPTION>`.

Кожний рядок таблиці починається з тегу **<TR>** і закінчується тегом **</TR>**. Якщо рядок містить заголовки стовпців таблиці, то використовують теги **<TH>** і **</TH>**, якщо ж дані, то **<TD>** і **</TD>**.

Приклад таблиці:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
  <title>Заголовок таблиці</title>
</head>
<body>
  <table border="1" width="100%" cellpadding="5">
    <tr>
      <th>Комірка 1</th>
      <th>Комірка 2</th>
    </tr>
    <tr>
      <td>Комірка 3</td>
      <td>Комірка 4</td>
    </tr>
  </table>
</body>
</html>
```

Основні атрибути:

- **align**

Задає вирівнювання таблиці по краю вікна браузера. Допустимі значення: **left** - вирівнювання таблиці по лівому краю, **center** - по центру і **right** - по правому краю. Коли використовуються значення **left** і **right**, текст починає обтікати таблицю збоку і знизу.

- **cellpadding**

Визначає відстань між межею комірки і її вмістом. Цей атрибут додає порожній простір до комірки, збільшуючи тим самим її розміри. Без **cellpadding** текст в таблиці «налипає» на рамку, погіршуючи тим самим його сприйняття. Додавання ж cellpadding дозволяє поліпшити читабельність тексту. При відсутності кордонів особливого значення цей атрибут не має, але може допомогти, коли потрібно встановити порожній проміжок між комірками.

- **cellspacing**

Задає відстань між зовнішніми межами комірок. Якщо встановлений атрибут border, товщина кордону приймається в розрахунок і входить в загальне значення.

- **cols**

Атрибут cols вказує кількість стовпців в таблиці, допомагаючи браузеру в підготовці до її відображення. Без цього атрибута таблиця буде показана тільки після того, як весь її вміст буде завантажено в браузер і проаналізовано. Використання атрибуту **cols** дозволяє дещо прискорити відображення вмісту таблиці.

- **width**

Задає ширину таблиці.

Встановлення ширини таблиці та комірок.

Ширина всіх комірок в одному стовпці однакова, а в одному рядку може бути різною. Ширину таблиці задають атрибутом WIDTH у тегу <TABLE>, а комірки – за допомогою того самого

атрибута у тегу <TD> або <TH>. Ширину таблиці можна визначати як у пікселях, так і у відсотках від ширини вікна, а комірки – у пікселях або у відсотках від ширини таблиці.

Об'єднання комірок.

Для об'єднання комірок використовують такі атрибути: **COLSPAN** (об'єднання по горизонталі, у рядку) і **ROWSPAN** (по вертикалі, у стовпці) тегу <TD>. Значеннями цих атрибутів є кількість об'єднаних стовпців або рядків. Наприклад, **COLSPAN=3** означає, що комірка розтягнута на 3 стовпці, а **ROWSPAN=2** – що комірка займає 2 рядки.

Форматування тексту та оформлення рамок таблиці.

Текст всередині комірок можна вирівнювати як по горизонталі, так і по вертикалі. Горизонтальне вирівнювання вмісту всього рядка задають за допомогою атрибута **ALIGN** у тегу <TR>; цей самий атрибут у тегах <TD> або <TH> в окремих комірках виконує таку саму роль. Атрибут **ALIGN** може набувати значень **center**, **right**, **left** або **justify**, які задають вирівнювання вмісту таблиці відповідно по центру, за правим чи лівим краєм або за шириною.

Приклади тегів <TR> та <TD> із використанням вирівнювання: <TR **ALIGN=right**>; <TD **ALIGN=center**>l. Алгебра</TD>. Вертикальне вирівнювання задають у тегах <TR>, <TD> або <TH> за допомогою атрибута **VALIGN**, який може набувати значень **top**, **bottom** та **middle**, що визначають вирівнювання відповідно за верхнім та нижнім краєм і по центру.

Для оформлення зовнішньої рамки таблиці можна

використовувати атрибут **FRAME** тегу **<TABLE>**. Його значення визначають, що браузер відображатиме:

- **box** – всі чотири сторони рамки;
- **above** – лише верхню межу рамки;
- **below** – лише нижню межу;
- **lhs** – лише ліву межу;
- **rhs** – лише праву межу;
- **hsides** – верхню й нижню межі рамки;
- **vsides** – ліву й праву межі рамки;
- **void** – зовнішня рамка не відображатиметься.

Відображення розділювальних ліній між стовпцями та рядками таблиці визначають за допомогою атрибута **RULES** тегу **<TABLE>**.

Він може набувати таких значень:

- **all** – відображати всі вертикальні та горизонтальні лінії;
- **rows** – лише горизонтальні лінії між рядками;
- **cols** – лише вертикальні лінії між стовпцями;
- **none** – не показувати розділювальних ліній.

Атрибут **BORDER** встановлює товщину кордону в пікселях навколо таблиці. При наявності цього атрибута також відображаються межі між комірками.

Для оформлення комірок таблиць за допомогою кольорів застосовують атрибут **BGCOLOR**. Колір можна визначати, вказуючи його назву англійською мовою або символ **#** та шістнадцяткове число.

Якщо потрібно задати колір для всієї таблиці, то атрибут **BGCOLOR** задають у тегу `<TABLE>`, для зміни кольору лише в одному рядку – в тегу `<TR>`, а налаштування кольору окремої комірки забезпечує наявність цього атрибута всередині тегу `<TD>` або `<TH>`.

Колір рамки таблиці можна задати за допомогою таких атрибутів:

- **BORDERCOLOR** – колір всієї рамки;
- **BORDERCOLORLIGHT** – колір світлої частини рамки;
- **BORDERCOLORDARK** – колір темної частини рамки.

Ці атрибути вставляють у тег `<TABLE>`. Щоб вони діяли, необхідна наявність ще й атрибута **BORDER**, який задає товщину рамки.

1.9. Зображення

Найбільш розповсюдженими в Інтернеті серед графічних форматів файлів є формати **GIF, JPG (JPEG) та PNG**.

Для задання фону сторінки у вигляді малюнка в тезі `<BODY>` використовують атрибут **BACKGROUND**, значенням якого є URL-адреса файлу зображення. Наприклад, тег `<BODY BACKGROUND="1. jpg">` визначає, що для фонового заповнення вебсторінки буде використано файл 1.jpg.

Приклад HTML-коду сторінки, в якій тло оформлене у вигляді малюнка, що міститься у файлі mone.jpg:

`<HTML>`

```
<TITLE>Моя фірма</TITLE>
<BODY BACKGROUND="none.jpg">
<H1><A HREF="1.html">Наша продукція</A></H1>
<H1><A HREF="2.html">Наші замовники</A></H1>
</BODY>
</HTML>
```

Для розміщення малюнків у HTML-документі використовують одинарний тег ****. Він має обов'язковий атрибут **SRC**, значенням якого є URL-адреса файлу зображення, записана в абсолютній (повністю) або відносній формі (починаючи від поточного каталогу; при цьому «батьківський» каталог позначають за допомогою двох крапок і слешу (. . /)). Під час відкривання документа браузер завантажує малюнок і відображає його в тому місці документа, де розташований тег ****.

Атрибути:

- **src** шлях до графічного файлу;
- **align** визначає як малюнок буде вирівнюватися по краю і спосіб обтікання текстом.

Атрибут **ALIGN** може набувати таких значень:

- ❖ **left** – зображення розміщене в лівій частині сторінки, текст обтікає його з правого боку;
- ❖ **right** – зображення розміщене у правій частині сторінки, текст обтікає його з лівого боку;
- ❖ **top** – обтікання немає, зображення розміщене в тексті, відповідний рядок якого вирівняно за верхньою межею малюнка;
- ❖ **bottom** – обтікання немає, зображення розміщене в тексті, відповідний рядок якого вирівняно за нижньою межею малюнка;

❖ **middle** – обтікання немає, зображення розміщене в тексті, відповідний рядок якого вирівняно по середній лінії малюнка.

За умовчанням атрибут **ALIGN** має значення **left**.

- **alt** - альтернативний текст для зображення, якщо воно не завантажиться.
- **border** - товщина рамки навколо зображення.

Графічний об'єкт буде показаний на вебсторінці у своїх реальних розмірах. Якщо є потреба у їх зміні (масштабуванні об'єкта), можна задати за допомогою атрибутів **WIDTH** та **HEIGHT**.

- **height** - висота зображення в пікселях.
- **width** - ширина зображення.

Для того щоб картинка не була розташована впритул до тексту, можна задати відступи. Це роблять за допомогою атрибута **HSPACE** для бокових полів та атрибута **VSPACE** для верхнього і нижнього полів. Значення аргументів задають у вигляді чисел, що визначають розміри відступів у пікселях.

- **hspace** – горизонтальний відступ від зображення до навколишнього контенту.
- **vspace** – вертикальний відступ від зображення до навколишнього контенту.

```
<HTML>
<TITLE>Малюнок</TITLE>
<BODY BACKGROUND="flamingo.jpg">
  <IMG SRC="mone.jpg" WIDTH=300 HEIGHT=200 ALT="Фламінго"
  align="middle" Hspace="10">
</BODY>
```

</HTML>

У прикладі, наведеному нижче, задано такі значення атрибутів, що малюнок розміщується ліворуч від тексту, який обтікатиме його. Поля навколо малюнка матимуть розмір 10 пікселів.

<HTML>

<TITLE>МАЛЮНОК</TITLE>

<BODY BACKGROUND="1.jpg">

Блакитні води океану та коралові рифи, ласкаве сонце і білосніжний прибережний пісок, фантастичний підводний світ і казкові птахи, буяння барв сходів і заходів, непрохідні ліси і п'яний запах орхідей...

Сприятливий клімат Острова Свободи дає можливість туристам відвідувати його цілий рік. Тут немає зайвої вологості, а спека не така страшна завдяки постійним подувам бризів із океану. Середньорічна температура повітря й води становить + 26–28°C.

</BODY>

</HTML>

1.10. Аудіо та відео на вебсторінках

Тег **<audio>** додає, відтворює і управляє настройками аудіозапису на вебсторінці. Шлях до файлу задається через атрибут **src** або вкладений тег **<source>**. Всередині контейнера **<audio>** можна написати текст, який буде виводитися в браузерах, які не працюють з цим тегом.

Синтаксис:

<audio src="URL"></audio>

Атрибути:

- **autoplay** - звук починає грати відразу після завантаження сторінки;

- **controls** - додає панель управління до аудіофайлу;

- **loop** - повторює відтворення звуку з початку після його завершення;

- **preload** - використовується для завантаження файлу разом із завантаженням вебсторінки;

- **src** - вказує шлях до воспроизводимому файлу.

Для універсального відтворення в зазначених браузерах аудіо кодують за допомогою різних кодеків і додають файли одночасно через тег **<source>**.

Синтаксис:

```
<audio><source src="URL"></audio>
```

Атрибути:

- **media** - визначає пристрій, для якого буде відтворюватися файл;

- **src** - адреса файлу;

- **type** - MIME-тип медійного джерела.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>audio</title>
  </head>
  <body>
    <p>Александр Клименков - Четырнадцать</p>
    <audio controls>
      <source src="music.ogg" type="audio/ogg">
    </audio>
  </body>
</html>
```

```
<source src="music.mp3" type="audio/mpeg">
  Тер audio не поддерживается вашим браузером.
  <a href="audio/music.mp3">Скачайте музыку</a>.
</audio>
</body>
</html>
```

Тег **<video>** додає, відтворює і управляє настройками відеоролика на вебсторінці. Шлях до файлу задається через атрибут **src** або вкладений тег **<source>**. Список підтримуваних браузерами аудіо та відеокодеків обмежений і наведено в табл. 1.

Для універсального відтворення в зазначених браузерах відео кодують за допомогою різних кодеків і додають файли одночасно через тег **<source>**.

Синтаксис:

```
<Video><Source src = "URL"></ Video>
```

Атрибути:

- **autoplay** - відео починає відтворюватися автоматично після завантаження сторінки;
- **controls** - додає панель управління до відеоролика;
- **height** - задає висоту області для відтворення відеоролика;
- **loop** - повторює відтворення відео з початку після його завершення;
- **poster** - вказує адресу картинки, яка буде відображатися, поки відео недоступне або не воспроизводится;
- **preload** - використовується для завантаження відео разом із завантаженням вебсторінки;

- **src** - вказує шлях до воспроизводимому відеоролику;
- **width** - задає ширину області для відтворення відеоролика.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>video</title>
  </head>
  <body>
    <video width="400" height="300" controls="controls" >
      <source src="duel.ogv" type='video/ogg'>
      <source src="duel.mp4" type='video/mp4'>
      <source src="duel.webm" type='video/webm'>
      Тер video не поддерживается вашим браузером.
      <a href="duel.mp4">Скачайте видео</a>.
    </video>
  </body>
</html>
```

1.11. Форми в HTML

Елемент **<form>** призначений для створення на вебсторінці форми. Форма використовується для обміну даними між користувачем і сервером. Застосування форм не обмежено відправкою даних на сервер, за допомогою клієнтських скриптів можна отримати доступ до будь-якого елементу форми. Документ може містити будь-яку кількість форм, але одночасно на сервер може бути відправлена тільки одна форма. Тому дані форм повинні бути незалежні.

Для відправки форми на сервер використовується кнопка **Submit**, або якщо натиснути клавішу Enter в межах форми. Якщо кнопка Submit відсутня на формі, то клавіша Enter імітує її використання, але тільки у випадку, коли у формі єдиний елемент **<input>**. Якщо таких елементів два і більше, натискання на **<Enter>** не дасть ніякого результату.

Коли форма відправляє дані на сервер, управління передається програмі, заданої атрибутом **action**. Попередньо браузер готує інформацію у вигляді пар «ім'я = значення», де ім'я визначається атрибутом **name** компонентів, вкладених в форму, а значення введено користувачем або встановлене в полі форми за замовчуванням.

Основні атрибути форми:

- **action** – вказує обробник, до якого звертаються дані форми при їх відправленні на сервер. Після виконання обробником дій по роботі з даними форми він повертає новий HTML-документ. Якщо

атрибут `action` відсутній, поточна сторінка перезавантажується, повертаючи всі елементи форми до їх значень по замовчуванню.

Синтаксис:

```
<form action="URL">  
...  
</form>
```

● **method** – повідомляє серверу про метод запиту. Може приймати одне із двох значень:

- ❖ **get** – цей метод є одним з найпоширеніших і призначений для отримання необхідної інформації та передачі даних в адресному рядку.

Адресний рядок приймає такий вигляд:

```
http://www.htmlbook.ru/handler.cgi?nick=%C2%E0%ED%FF+%D8  
%E0%EF%EE%F7%EA%E8%ED&page=5
```

Параметри перераховуються після знаку питання, зазначеного після адреси CGI-програми і розділяються між собою символом амперсанда (&). Нелатинські символи перетворюються в шістнадцяткове представлення (у формі %HH, де HH – шістнадцятковий код для значення ASCII-символа), пробіл замінюється на плюс (+). Зручність використання цього методу полягає в тому, що адреса з усіма параметрами можна використовувати неодноразово, зберігши його, наприклад, в закладки браузера, а також міняти значення параметрів прямо в адресному рядку;

❖ метод **post** посилає на сервер дані в запиті браузера. Це дозволяє відправляти більшу кількість даних, ніж доступно методом **get**, оскільки у нього встановлено обмеження в 4 Кб. Великі обсяги даних використовуються в форумах, поштових службах, заповненні бази даних, при пересиланні файлів та ін.

● **target** – визначає вікно, в яке буде завантажуватися сторінка. Якщо **target** не встановлений, повернутий результат показується в поточному вікні. Як значення використовується ім'я вікна або фрейму, що задане атрибутом **name**. Якщо встановлено неіснуюче ім'я, то буде відкрито нове вікно. В якості зарезервованих імен можна вказувати наступні:

_blank – завантажує сторінку в нове вікно браузера;

_self – завантажує сторінку в поточне вікно;

_parent – завантажує сторінку у батьківський фрейм, якщо фреймів немає, то працює як **_self**;

_top – скасовує всі фрейми і завантажує в повне вікно, якщо фреймів немає, то працює як **_self**;

● **name** – визначає унікальне ім'я форми. Як правило, ім'я форми використовується для доступу до її елементів за допомогою скриптів;

● **novalidate** – скасовує вбудовану перевірку даних введених користувачем у формі на коректність. Така перевірка здійснюється браузером автоматично при відправці форми на сервер і

відбувається для полів `<input type = "email">`, `<input type = "url">`, а також за наявності атрибуту `pattern` або `required`. Синтаксис:

`<form novalidate>...</form>`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Тер FORM, атрибут novalidate</title>
  </head>
  <body>
    <form action="handler.php" novalidate>
      <p><input required placeholder="Имя"></p>
      <p><input type="email" required placeholder="Ваш
email"></p>
      <p><input type="url" required placeholder="Сайт"></p>
      <p><input type="submit"></p>
    </form>
  </body>
</html>
```

● **enctype** – визначає спосіб кодування даних форми при їх відправленні на сервер. Зазвичай встановлювати значення атрибута `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**. Синтаксис:

`<form enctype = "application / x-www-form-urlencoded | multipart / form-data | text / plain">`

...

`</ form>`

Значення атрибуту **enctype**:

❖ **application / x-www-form-urlencoded** – замість прогалин ставиться +, символи українських букв кодуються їх шістнадцятковими значеннями (наприклад, %D0%90%D0%BD%D1%8F замість Аня) - (за замовчуванням);

❖ **multipart / form-data** – дані не кодуються. Це значення застосовується при відправці файлів;

❖ **text / plain** – прогалини замінюються знаком +, літери та інші символи не кодують;

● **autocomplete** – керує автозаповненням полів форм. Значення може бути перекрито атрибутом autocomplete у конкретних елементів форми. Автозаповнення виконує браузер, який запам'ятовує написання при першому введенні значення, а потім підставляє їх при повторному наборі в поля форми. При цьому автозаповнення може бути відключено в налаштуваннях браузера і не може бути в такому випадку змінено за допомогою атрибуту autocomplete. Синтаксис:

<form autocomplete="on | off">...</form> (on за замовчуванням)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ter form, атрибут autocomplete</title>
  </head>
  <body>
    <form autocomplete="on">
      <p>Имя: <input name="user"></p>
      <p>Пароль: <input name="pass" type="password"></p>
```

```
<p><input type="submit" value="Отправитъ"></p>
</form>
</body>
</html>
```

- **accept-charset** Синтаксис:

<form action="URL" accept-charset="кодировка">...</form>

Назва кодування, наприклад Windows-1251, UTF-8 та ін.

Елемент **<input>** дозволяє створювати текстові поля, кнопки, перемикачі, прапорці та ін.. Хоча елемент **<input>** не обов'язково поміщати всередину контейнера **<form>**, але якщо введені користувачем дані повинні бути відправлені на сервер, для обробки, то вказувати **<form>** обов'язково.

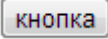
Атрибути:

- **name** – ім'я поля, призначене для того, щоб обробник форми міг його ідентифікувати. Синтаксис:

<input type="..." name="ім'я">

- **type** – визначає вид елемента. Він дозволяє задавати різні елементи форми. Синтаксис:

<input type="button|checkbox|file|hidden|image|password|radio|reset|submit|text">

- ❖ **button** – кнопка,  . Використовується для запуску певної дії. На відміну від інших елементів, параметр **value** використовується не для задня значення що повертається при натисканні, а для формування напису на самій кнопці.

Дія, що виконується при натисканні кнопки, вказується за допомогою атрибуту `onClick`;

❖ **checkbox** – прапорець,  `варіант1`  `варіант2`  `варіант3` .

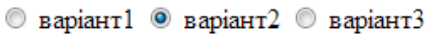
Дозволяє вибрати кілька варіантів із запропонованих, або відповісти на певне запитання так чи ні. Якщо елемент вибраний то обробнику буде передано значення із `value`, в іншому випадку не буде передано нічого, тобто не вибрані прапорці взагалі ніяк не проявляють себе в переданому наборі даних. Щоб елемент був відзначений по замовчуванню необхідно позначити його атрибутом `checked`. Як і перемикач, не має стандартних можливостей для виведення пояснюючого тексту;

❖ **file** – поле для введення імені файлу, який пересилається на сервер;

❖ **hidden** – приховане поле. Воно ніяк не відображається на вебсторінці;

❖ **image** – кнопка із зображенням. Як і звичайна кнопка, використовується для запуску певної дії. Може містити будь-яке зображення, при натисканні на яке виконується вказана дія. Атрибут `src` вказує шлях до файлу, що містить зображення;

❖ **password** – поле для паролів,  . Аналогічне текстовому полю, за винятком того що символи, що вводить користувач, приховуються;

- ❖ **radio** – перемикач , призначений для вибору одного варіанту з кількох можливих. Атрибут `value` задає значення. Якщо користувач вибере перемикач, то обробнику буде передано саме це значення. При необхідності можна вказати параметр **checked**, який вказує на те, що перемикач відзначений при завантаженні сторінки. Перемикачі можна об'єднувати в групи, для цього вони повинні мати одне й теж ім'я, вказане в атрибуті `name`. Можна вибрати тільки один перемикач із групи. Якщо в перемикачів різні імена то кожен із них утворює окрему групу і можна вибрати кожен із них. Та на відміну від прапорця відмінити свій вибір вже неможливо. Перемикач не має стандартних можливостей для виведення пояснюючого тексту, тому його зазвичай поміщають в інший елемент, що містить текст, наприклад мітку чи параграф;
- ❖ **reset** – кнопка для очищення форми. При натисканні всім елементам форми повертаються значення по замовчуванню, а дані введені користувачем, втрачаються;
- ❖ **submit** – кнопка для відправки даних із форми. При натисканні відбувається відправка даних на сервер. Текст на кнопці залежить від установок браузера та може бути змінено за допомогою атрибуту `value`. При заданому параметрі `onClick` спочатку виконуються вказані там дії, а

потім, все рівно, відбувається відправка результатів на сервер;

❖ **text** – текстове поле, , застосовується для введення користувачем певної інформації. Основні атрибути: розмір поля `size` та максимальна допустима кількість символів `maxlength`. Якщо вказано атрибут `value`, то поле буде містити даний текст як значення по замовчуванню.

Елемент **<button>** призначений для створення створює кнопки і нагадує `<input>` з атрибутом `type = "button | reset | submit"`. На відміну від нього, `<button>` пропонує розширені можливості по створенню кнопок. Тег парним, і на подібній кнопці можна розміщувати будь-які елементи HTML, у тому числі зображення. Використовуючи стилі можна визначити вид кнопки шляхом зміни шрифту, кольору фону, розмірів та інших параметрів.

Основні атрибути:

- `button` – звичайна кнопка;
 - `reset` – кнопка для очищення форми;
 - `submit` – кнопка для відправки даних форми на сервер.
- Значення по замовчуванню - `submit`.

2. CSS

2.1. Основи CSS

Стиль – це набір правил оформлення та форматування, який можна застосувати до різних елементів веб-документа.

Каскадні таблиці стилів (CSS, Cascading Style Sheets) містять параметри форматування частини або всього тексту вебсторінки. Таблиці каскадних стилів дають змогу визначити єдиний стиль оформлення для різних сторінок документа і швидко модифікувати його зміною відповідного параметра у таблиці стилів.

Параметрів форматування, які можна задавати за допомогою стилів, досить багато. Це, зокрема:

- background – колір тла;
- font-family – стиль шрифту (гарнітура);
- font-size – розмір шрифту;
- font-weight – жирність шрифту;
- color – колір шрифту;
- text-decoration – оздоблення тексту;
- text-align – вирівнювання тексту;
- margin-top – відступ від верхнього рядка абзацу;
- line-height – міжрядкова відстань.

Є три способи зв'язку каскадних стилів із HTML-документом:

1. Підключення зовнішньої таблиці стилів (пов'язані стилі).

2. Розташування опису стилів у розділі HEAD документа (глобальні стилі).

3. Задання властивостей стилів безпосередньо в тегах абзаців чи заголовків (вбудовані стилі).

1. Створення та підключення зовнішньої таблиці стилів

Зовнішня таблиця стилів (External Style Sheet) – це текстовий файл із розширенням .css. Його підключають до HTML-документа за допомогою тегу **<LINK>**, який записують у розділі **<HEAD>**, наприклад:

<link rel="stylesheet" type="text/css" href="адреса файлу">

Значення атрибутів rel і type залишаються незмінними незалежно від коду, як наведено в даному прикладі. Значення href задає шлях до CSS-файлу, він може бути заданий як відносно, так і абсолютно. Зауважимо, що таким чином можна підключати таблицю стилів, яка знаходиться на іншому сайті. Файл зі стилем не зберігає ніяких даних, крім синтаксису CSS. Підключення зовнішньої таблиці стилів:

<link rel="stylesheet" href="mysite.css">

Файл зі стилем

```
H3 {
    color: #000080;
    font-size: 200%;
    font-family: Arial, Verdana, sans-serif;
    text-align: center;
}
P {
    padding-left: 20px;
}
```

2. Розташування опису стилів у розділі HEAD документа у блоці, який обмежений тегамі <style> та </style>

```
<head>
<style>
  Тер1 {властивість 11 : значення 11; властивість 12 : значення
12; ...; властивість 1n : значення 1n }
  Тер2 {властивість 21 : значення 21; властивість 22 : значення
22; ...; властивість 2m : значення 2m}
</style>
</head>
```

Найпростіша внутрішня таблиця стилів – це послідовність значень тегів, кожне з яких записується, як правило, з нового рядка. Визначення тегу містить його ім'я без кутових дужок, за яким у фігурних дужках через крапку з комою перелічують властивості тегів та їхні значення, розділені двокрапками.

3. Вбудовані (внутрішні) стилі (Inline Styles) вставляють в теги (теги заголовків <h1>...<h6>, абзацу <p>, тіла <body>, а також у теги <div>, тощо) за допомогою атрибута style, в якому перелічують властивості та їх значення. Наприклад:

<p style="font-size: 48pt; color: yellow">

Визначені у такий спосіб властивості мають найвищий пріоритет порівняно з іншими, оскільки вони визначені безпосередньо у тегу.

Цей підхід використовують для оформлення невеликої кількості елементів.

<h2 style="font-size: 48pt; font-family: Arial">Текст...</h2>

Приклад:

```
<!DOCTYPE html>
```

```

<html>
  <head>
    <link rel="stylesheet" href="mysite.css">
    <style>
      h1 {font-size: 38pt; color: red}
      h2 {font-size: 20pt; color: blue}
    </style>
    <title>Приклад використання CSS</title>
  </head>
  <body>
    <h1>Для заголовка першого рівня визначено розмір 38 pt,
а колір тексту - червоний </h1>
    <h2>Для заголовка другого рівня визначено розмір 20 pt,
а колір тексту - синій </h2>
    <h3>Для заголовка 3 рівня визначено з файла mysite.css
</h3>
    <h4>Для заголовка 4 рівня визначено з файла mysite.css
</h4>
    <h2 style="font-size: 48pt; font-family: Arial; color:
yellow"> Текст вбудованого стилю (внутрішнього)</h2>
    <p>Для цього абзацу стиль не застосовано, для оформлення
тексту використано атрибути за умовчанням.</p>
  </body>
</html>

```

Імпорт CSS.

У поточну стильову таблицю можна імпортувати вміст CSS-файлу за допомогою команди `@import`. Цей метод допускається використовувати спільно з пов'язаними або глобальними стилями (зовнішні таблиці стилів), але ніяк не з внутрішніми стилями. Загальний синтаксис наступний:

`import url ("ім'я файлу") типи носіїв;`

`import "ім'я файлу" типи носіїв;`

Після ключового слова **@import** вказується шлях до стилісового файлу одним з двох наведених способів - за допомогою **url** або без нього.

```
@import url("style/header.css");
```

```
@import "/style/main.css" screen; /* Стиль для виведення  
результату на монітор */
```

```
@import "/style/smart.css" print, handheld; /* Стиль для друку і  
смартфона */
```

Типи носіїв.

У CSS для вказівки типу носіїв застосовуються команди **@import** і **@media**, за допомогою яких можна визначити стиль для елементів залежно від того, виводиться документ на екран або на принтер.

- **all** – всі типи. Це значення використовується за умовчанням;
- **aural** – мовні синтезатори, а також програми для відтворення тексту вголос. Сюди, наприклад, можна віднести мовні браузерери;
- **braille** – пристрої, засновані на системі Брайля, які призначені для сліпих людей;
- **handheld** – КПК та аналогічні їм апарати;
- **print** – друкуючі пристрої на кшталт принтера;
- **projection** – проектор;
- **screen** – екран монітора;
- **tv** – телевізор.

Команда `@media` дозволяє вказати тип носія для глобальних або пов'язаних стилів і в загальному випадку має наступний синтаксис:

```
@media тип носія 1 {  
    Опис стилю для типу носія 1  
}  
  
@media тип носія 2 {  
    Опис стилю для типу носія 2  
}
```

2.2. Синтаксис CSS

Таблиця стилів складається з набору правил. Кожне правило, у свою чергу, складається з одного або декількох селекторів і блоку визначень, що відносяться до них. Загальний спосіб запису:

Селектор{властивість1: значення; властивість2: значення;...; властивістьN: значення;}

Селектор – це деяке ім'я стилю, для якого додаються параметри форматування. В якості селектора виступають теги, класи та ідентифікатори.

Спочатку пишеться ім'я селектора, наприклад, `<table>`, це означає, що всі стильові параметри будуть застосовуватися до тегу `<table>`, потім йдуть фігурні дужки, в яких записується визначення властивостей. Визначення властивостей складається із стильової властивості і її значення, що вказується після двокрапки.

Визначення властивостей розділяються між собою крапкою з комою, в кінці цей символ можна опустити.

CSS не чутливий до регістру, перенесення рядків, пробілів і символів табуляції, тому форма запису залежить від бажання розробника.

Приклад:

Нижче описані ідентичні стилі.

```
h1 { color: #a6780a; font-weight: normal; border-bottom:
2px solid black}

H1 {

    border-bottom: 2px solid black
    color: #a6780a;
    font-weight: normal;

}
```

Для селектора допускається додавати кожне правило окремо:

```
td {background: olive;}
td {color: white;}
td {border: 1px solid black;}
```

Компактна форма запису:

```
td {
    background: olive;
    color: white;
    border: 1px solid black;
}
```

Якщо для селектора спочатку задається властивість з одним значенням, а потім та ж властивість, але вже з іншим значенням, то застосовуватися буде те значення, яке в коді встановлено нижче

```
p {color: green;}
p {color: red;}
```

У даному прикладі для селектора `p` колір тексту спочатку задається зеленим, а потім червоним. Оскільки значення `red` розташоване нижче, то воно в підсумку і буде застосовуватися до тексту.

Щоб позначити, що текст є коментарем, застосовують наступну конструкцію `/* ...коментар... */`

Приклад:

```
/* Мій стиль */
div {
    width: 200px; /* Ширина блоку */
    margin: 10px; /* Поля навколо елемента */
    float: left; /* Обтікання по правому краю */
}
```

Коментарі можна додавати в будь-яке місце CSS-документа, а також писати текст коментаря у кілька рядків. Вкладені коментарі неприпустимі.

Значення стильових властивостей.

Все різні значення стильових властивостей можуть бути зведені до певного типу: рядок, число, відсотки, розмір, колір, адресу або ключове слово.

Рядки. Будь-які рядки необхідно брати в подвійні або одинарні лапки. Якщо всередині рядка потрібно використати лапки чи слеш, то необхідно додати перед ними слеш (екранувати символ).

Приклад: "Готель 'Турист'", або "Готель \" Турист \""

Числа. Значенням може виступати ціле число, що містить цифри від 0 до 9 і десятковий дріб, в якій ціла і десяткова частина розділяються крапкою.

Приклад: 500, 3.14

Відсотки. Процентний запис зазвичай застосовується в тих випадках, коли треба змінити значення щодо батьківського елемента або коли розміри залежать від зовнішніх умов. Так, ширина таблиці 100% означає, що вона буде підлаштовуватися під розміри вікна браузера і змінюватися разом з шириною вікна

Розміри. Для завдання розмірів різних елементів, в CSS використовуються **абсолютні і відносні одиниці виміру**. **Абсолютні одиниці** не залежать від пристрою виводу, а відносні одиниці визначають розмір елемента щодо значення іншого розміру. **Відносні одиниці** зазвичай використовують для роботи з текстом, або коли треба обчислити співвідношення між елементами.

em	Розмір шрифту поточного елемента
ex	Висота символу x
px	Піксель
%	Відсоток

Одиниця **em** це змінюване значення, яке залежить від розміру шрифту поточного елемента (розмір встановлюється через стильову властивість font-size). В кожному браузері закладений розмір тексту, застосовуваний в тому випадку, коли цей розмір явно не заданий. Тому спочатку 1em дорівнює розміру шрифту, заданого в браузері за замовчуванням або розміром шрифту батьківського елемента. Відсотковий запис ідентична **em**, в тому сенсі, що значення **1em і 100% рівні**.

Одиниця **ex** визначається як висота символу «x» в нижньому регістрі. На **ex** розповсюджуються ті ж правила, що і для **em**, а саме, він прив'язаний до розміру шрифту, заданого в браузері за замовчуванням, або до розміру шрифту батьківського елемента.

Піксель – це точка певного кольору, що відображається монітором або іншим подібним пристроєм, наприклад, смартфоном. Розмір пікселя залежить від розширення пристрою і його розмірів.

Відсотковий запис зазвичай застосовується в тих випадках, коли треба змінити значення щодо батьківського елемента або коли розміри залежать від зовнішніх умов. Так, ширина таблиці 90% значить, що вона буде підлаштовуватися під розміри вікна і змінюватися разом з шириною вікна.

Абсолютні одиниці застосовуються рідше, ніж відносні і зазвичай при роботі з текстом.

in	Дюйм (1 дюйм дорівнює 2,54 см)
cm	Сантиметр
mm	Міліметр
pt	Пункт (1 пункт дорівнює 1/72 дюйма)
pc	Піка (1 піка дорівнює 12 пунктам)

Найпоширенішою одиницею є пункт, який використовується для вказівки розміру шрифту.

Адреси (URI, Uniform Resource Identifiers, уніфікований ідентифікатор ресурсів) застосовуються для вказівки шляху до файлу, наприклад, для установки фонові картини на сторінці.

Для цього застосовується ключове слово `url ()`, всередині дужок пишеться відносний або абсолютний адресу файлу. При цьому адреса можна задавати в необов'язкових подвійних лапках.

Колір в стилях можна задавати трьома способами: за шістнадцятковим значенням, за назвою і в форматі RGB.

1. За шістнадцятковим значенням.

Для завдання кольорів використовуються числа в шістнадцятковому коді. Шістнадцяткова система, на відміну від десяткової системи, базується, як випливає з її назви, на числі 16. Цифри будуть наступні: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Цифри від 10 до 15 замінені латинськими літерами. Числа більше 15 в шістнадцятковій системі утворюються об'єднанням двох чисел в одне. Наприклад, числу 255 в десятковій системі відповідає число FF в шістнадцятковій системі. Щоб не виникало плутанини у визначенні системи числення, перед шістнадцятковим числом ставлять символ решітки #, наприклад #666999. Кожен з трьох кольорів - червоний, зелений і синій - може приймати значення від 00 до FF. Таким чином, позначення кольору розбивається на три складові `#rrggbb`, де перші два символи відзначають червону компоненту кольору, два середніх - зелену, а два останніх - синю. Допускається використовувати скорочену форму виду `#rgb`, де кожен символ слід подвоювати (`#rrggbb`). Наприклад, запис `#fe0` розцінюється як `#ffee00`.

2. За назвою.

Браузери підтримують деякі кольори по їх назві:

- **white** #ffffff або #fff - білий;
- **silver** # c0c0c0 - сірий;
- **gray** # 808080 - темно-сірий;
- **black** # 000000 або # 000 - чорний;
- **maroon** # 800000 - темно-червоний;
- **red** # ff0000 або # f00 - червоний;
- **orange** # ffa500 - помаранчевий;
- **yellow** # ffff00 або # ff0 - жовтий;
- **olive** # 808000 - оливковий;
- **lime** # 00ff00 або # 0f0 - світло-зелений;
- **green** # 008000 - зелений;
- **aqua** # 00ffff або # 0ff - блакитний;
- **blue** # 0000ff або # 00f - синій;
- **navy** # 000080 - темно-синій;
- **teal** # 008080 - синьо-зелений;
- **fuchsia** # ff00ff або # f0f - рожевий;
- **purple** # 800080 - фіолетовий.

3. В форматі RGB.

Можна визначити колір, використовуючи значення червоною, зеленою і синій складової в десятковому численні. Значення кожного з трьох кольорів може приймати значення від 0 до 255. Також можна задавати колір у відсотковому відношенні. Спочатку вказується ключове слово **rgb**, а потім в дужках, вказуються компоненти кольору, наприклад **rgb (255, 51, 51)** або **rgb (100%, 20%, 20%)**.

В якості значень активно застосовуються **ключові слова**, які визначають бажаний результат дії стилевих властивостей. Ключові слова пишуться без лапок.

2.3. Селектори

В якості селектора може виступати будь-який тег HTML для якого визначаються правила форматування, такі як: колір, фон, розмір і т.д. Ім'я тега, не чутливе до регістра.

Тег {властивість 1: значення; властивість 2: значення; ...}

P {text-align: justify; color: green;}

У даному прикладі змінюється колір тексту і вирівнювання тексту абзацу. Стил буде застосовуватися тільки до тексту, який розташовується усередині контейнера <p>. Хоча стиль можна застосувати до будь тегу, результат буде помітний тільки для тегів, які безпосередньо відображаються в контейнері <body>.

Щоб встановити одночасно один стиль для всіх елементів вебсторінки, наприклад, задати шрифт або накреслення тексту використовують **універсальний селектор, який відповідає будь-якому елементу вебсторінки**. Для позначення універсального селектора застосовується символ зірочки (*) і в загальному випадку синтаксис буде наступний. *{Опис правил стилю}

Класи.

Класи застосовують, коли необхідно визначити стиль для індивідуального елемента сторінки або задати різні стилі для

одного тега. Усередині стилю спочатку пишеться бажаний тег, а потім, через точку користувальницьке ім'я класу (Тег.Ім'я_класу).

Тег.Ім'я класу {властивість 1: значення; властивість 2: значення; ...}

Імена класів повинні починатися з латинської літери і можуть містити в собі символ дефіса (-) та підкреслення (_). Використання російських букв в іменах класів неприпустимо. Щоб вказати в коді HTML, що тег використовується з певним класом, до тегу додається атрибут **class = "Ім'я_класу"**.

Можна, також, використовувати класи і без вказівки тега. Синтаксис в цьому випадку буде наступний: **.Ім'я_класу{...}**. При такому записі, клас можна застосовувати до будь-якого тегу.

```
.gost {  
    color: green; /* Цвет текста */  
    font-weight: bold; /* Жирное начертание */  
}
```

До будь-якого тегу одночасно можна додати кілька класів, перераховуючи їх в атрибуті class через пробіл. У цьому випадку до елементу застосовується стиль, описаний в правилах для кожного класу. Оскільки при додаванні декількох класів вони можуть містити однакові стильові властивості, але з різними значеннями, то береться значення класу, який описаний в коді нижче.

Ідентифікатор.

Ідентифікатор («ID селектор») визначає унікальне ім'я елемента, яке використовується для зміни його стилю і звернення до нього через скрипти.

При описі ідентифікатора спочатку вказується символ решітки (#), потім йде ім'я ідентифікатора. Воно повинно починатися з латинської символу і може містити в собі символ дефіса (-) та підкреслення (_). Використання російських букв в іменах ідентифікатора неприпустимо.

#Ім'я ідентифікатора {властивість 1: значення; властивість 2: значення; ...}

```
#gost {  
    color: green; /* Цвет текста */  
    font-weight: bold; /* Жирное начертание */  
}
```

Звернення до ідентифікатора відбувається аналогічно класам, але використовується атрибут **id**, значенням якого виступає ім'я ідентифікатора. Символ решітки при цьому вже не вказується.

<div id="gost">

Ідентифікатори можна застосовувати до конкретного тегу.

Тег #Ім'я ідентифікатора {властивість 1: значення; властивість 2: значення; ...}

Спочатку вказується ім'я тега, потім без пробілів символ решітки та назва ідентифікатора.

Періодично піднімається суперечка про доцільність використання ідентифікаторів у верстці. Насправді немає різниці,

через що задавати стилі, але слід пам'ятати про деякі особливості ідентифікаторів і класів:

- у коді документа кожен ідентифікатор унікальний і повинен бути включений лише один раз;
- ім'я ідентифікатора чутливе до регістру;
- стиль для ідентифікатора має пріоритет вище, ніж у класів;
- клас може використовуватися в коді неодноразово для різних елементів;
- імена класів чутливі до регістра;
- класи можна комбінувати між собою, додаючи кілька класів до одного тегу.

Контекстний селектор складається з простих селекторів розділених пропуском. Синтаксис:

1. Стиль автора з додаванням !important.
2. Стиль користувача.
3. Стиль автора.

3. СИСТЕМА КОНТРОЛЮ ВЕРСІЙ GIT

3.1 Установка Git

Git – це система контролю версій (**Version Control System, VCS**), яка дозволяє відстежувати зміни у вашому коді, працювати в команді без ризику втрати роботи та швидко повертатися до попередніх версій проекту.

GitHub – це сервіс, що забезпечує зберігання коду та даних в репозиторіях на основі Git, пропонуючи як безкоштовні, так і платні плани.

GitHub дозволяє:

- Зберігати відкриті (до 300 Мб безкоштовно) або закриті репозиторії.
- Додавати права на запис для інших користувачів.
- Автоматизувати інтеграцію та деплой за допомогою CI/CD.

Установка Git для Windows

1. Перейдіть на офіційний сайт Git <https://git-scm.com/>
2. Завантажте інсталятор для Windows.
3. Запустіть інсталятор та дотримуйтесь інструкцій:
 - a. Виберіть **"Run Git from the Windows Command Prompt"**.
 - b. Інші налаштування залиште за замовчуванням.

4. Перезавантажте систему або завершіть сеанс користувача і знову увійти, щоб застосували зміни до системної змінної PATH.
5. Перевірте встановлення командою: `git --version`.

Установка Git для Linux

На більшості дистрибутивів **Linux Git** можна встановити через стандартний менеджер пакетів. Ось інструкції для найпоширеніших систем:

Ubuntu / Debian

Виконайте команду:

```
sudo apt update  
sudo apt install git
```

Перевірте встановлення:

```
git -version
```

Fedora

Виконайте команду:

```
sudo dnf install git
```

Перевірте встановлення:

```
git -version
```

Arch Linux / Manjaro

Виконайте команду:

```
sudo pacman -S git
```

Перевірте встановлення:

```
git -version
```

3.2 Налаштування SSH-ключів

Перед тим як зареєструватися на **GitHub**, спершу потрібно згенерувати **SSH-ключ**. Він необхідний для швидкого підключення до **GitHub** без введення пароля. Без **SSH-ключа** ви не зможете повноцінно працювати з репозиторіями.

Створення SSH-ключів у Windows:

1. Запустіть **Git Bash**:

```
ssh-keygen -t rsa -C "ваш_email@example.com"
```

2. Усі запити під час генерації залишайте порожніми,

натискаючи **Enter**.

3. Ключі будуть збережені у: `C:\Users\<ВашКористувач>\.ssh`

Створення SSH-ключів у Linux:

1. Перевірте наявність ключів у папці `ls ~/.ssh`:

2. Якщо є файли `id_rsa` і `id_rsa.pub`, вони вже згенеровані.

3. Для створення нового ключа:

```
ssh-keygen -t rsa -C ваш\_email@example.com
```

4. Введіть свою електронну пошту, потім натисніть **Enter** для підтвердження місця збереження ключа.

Створення SSH-ключів у macOS:

1. Перевірте наявність ключів у папці `ls ~/.ssh`:

2. Якщо є файли `id_rsa` і `id_rsa.pub`, вони вже згенеровані.

3. Для створення нового ключа:

```
ssh-keygen -t rsa -C ваш_email@example.com
```

4. Введіть свою електронну пошту, потім натисніть **Enter** для підтвердження місця збереження ключа.

3.3 Реєстрація на GitHub та настройка аккаунта та репозиторії

Ось покрокова інструкція реєстрації на **GitHub**:

1. Перейдіть на офіційну сторінку GitHub: <https://github.com>.
2. У правому верхньому куті сторінки натисніть кнопку "Sign up", щоб почати процес реєстрації.
3. Уведіть у спеціальне поле ім'я користувача, яке буде використовуватися на **GitHub**.
4. Вкажіть дійсну електронну пошту, на яку GitHub надішле повідомлення для підтвердження.
5. Введіть надійний пароль.
6. GitHub пропонує кілька планів, включаючи безкоштовний варіант. Для початківців достатньо вибрати безкоштовний план "Free", який надає базові функції, такі як безкоштовні приватні репозиторії.
7. Далі, **GitHub** попросить вас підтвердити, що ви не робот.
8. Після цього GitHub надішле вам електронний лист з посиланням для підтвердження реєстрації. Перейдіть за посиланням, щоб завершити створення облікового запису.
9. **Готово!**

Тепер у вас є обліковий запис на GitHub, і ви можете почати працювати з репозиторіями, створювати нові проєкти або приєднуватися до існуючих.

Як встановити SSH-ключ

1. Перейдіть в **Settings** на GitHub (використовуючи іконку профілю в правому верхньому куті).
2. У меню зліва виберіть **"SSH and GPG keys"**, натисніть *"New SSH key"*.
3. В полі **"Title"** вкажіть ім'я ключа (наприклад, комп'ютер або пристрій).
4. В полі **"Key"** вставте вміст файлу `id_rsa.pub` (відкрийте його в текстовому редакторі).
5. Натисніть *"Add SSH key"*.
6. Перевірте з'єднання за допомогою команди:

```
ssh -T git@github.com
```

3.4 Створення репозиторію

Першим кроком є створення репозиторію для вашого проєкту. Для цього є два способи:

1. **Спосіб 1:** Клацнути правою кнопкою миші в порожньому місці в папці `git` та вибрати `Git init`.

2. **Спосіб 2:** Відкрити `Git Bash` і виконати команду:

```
git init
```

Після цього в папці з'явиться прихована папка **.git**, яка міститиме налаштування репозиторію.

Налаштування глобальних параметрів користувача

Git дозволяє працювати з різними користувачами для різних репозиторіїв. Для того, щоб правильно ідентифікувати коміти, необхідно налаштувати ваше ім'я та електронну пошту.

Виконайте наступні команди в **Git Bash**:

1. Для налаштування імені користувача:

```
git config --global user.name "Ваше ім'я"
```

2. Для налаштування email користувача:

```
git config --global user.email "Ваш e-mail"
```

Після того, як ви **налаштували ім'я та email**, можна перевірити, чи правильно застосувались налаштування. Для цього виконайте команду:

```
git config -list
```

Виведеться список ваших налаштувань, де ви зможете побачити **user.name i user.email**.

3.5 Git: ігнорування файлів

Перед додаванням файлів у систему контролю версій необхідно вказати **Git**, які файли потрібно ігнорувати. Наприклад, у вашому проєкті можуть бути файли логів або тимчасові файли, які не потрібно відслідковувати. Для цього потрібно створити файл **.gitignore** в корені проєкту, відкрити файл для редагування в текстовому редакторі, додати правила до **.gitignore**, тобто вказати файли та папки, які повинні бути проігноровані. Наприклад:

а. Для ігнорування всієї папки logs, додаємо таке правило:

```
Logs/
```

Це означає, що вся папка logs буде проігнорована **Git**.

б. Для ігнорування всіх текстових файлів (.txt) у папці Docs, додаємо таке правило:

```
Docs/*.txt
```

Це дозволяє ігнорувати тільки файли з розширенням `.txt` у папці `Docs`, але інші файли в цій папці будуть додаватися до репозиторію.

с. Коментарі у файлі `.gitignore` починаються з символу `#`.

Наприклад:

```
# Ігноруємо папку logs
logs/
```

Після налаштування **.gitignore** можна перевірити, як **Git** бачить файли в проєкті. Для цього виконайте команду:

```
git status
```

Дана команда покаже так звані **untracked files** - файли, які не під контролем **Git**. Файли та папки які у нас в ігнорі ця команда не покаже. Так само не покаже файли, які знаходяться в папках.

Виконаємо цю команду, але з додатковим ключем:

```
git status --untracked-files-all
```

За допомогою цього додаткового ключа **гіт** нам покаже ВСІ файли, включаючи ті, що знаходяться в папках.

Для перевірки статусу файлів у конкретній папці (наприклад, в папці `Docs`), використовуйте команду:

```
git status Docs
```

Git покаже, які файли в цій папці будуть додані до репозиторію, з урахуванням налаштувань в **.gitignore**.

3.6 Додавання та видалення файлів в індексі Git

Перед будь-якими маніпуляціями завжди корисно перевірити статус репозиторію, щоб зрозуміти, що відбувається з файлами. Для цього використовуємо команду:

```
git status
```

Дана команда покаже список файлів, які не відслідковуються Git (untracked files), тобто файли, що ще не додані до контролю версій. Для того щоб додати всі файли в репозиторій, можна використовувати команду:

```
git add .
```

Ця команда додасть всі файли, за винятком тих, що вказані в .gitignore, в індекс Git. Після виконання цієї команди перевіряємо статус репозиторію:

```
git status
```

Файли, що раніше мали червоний колір - **untracked** змінили свій колір на зелений – тепер файли готові до коміту.

Якщо ви випадково додали файл, який не хочете відслідковувати (наприклад, забули додати його до .gitignore), ви можете видалити його з індексу Git, використовуючи команду:

```
git rm --cached <ім'я_файлу>
```

Ця команда видаляє файл тільки з індексу, але не з фізичної файлової системи.

3.7 Робота з файлами в Git: додавання, індексація та коміти

Індексація – це процес підготовки файлів до коміту в **Git**. Коли ви змінюєте файли, **Git** не одразу зберігає ці зміни в історії. Спочатку ви повинні додати файли в індекс, щоб сказати **Git**, які зміни ви хочете зберегти. Тільки після цього можна зробити коміт, щоб ці зміни потрапили в історію репозиторію.

Якщо ми хочемо додати файл, наприклад "index.html", використовуємо команду:

```
git add index.html
```

Тепер файл доданий до індексу, а також, якщо файл було змінено, він буде позначений як готовий до коміту(зелений колір).

Коли ми змінюємо файли (наприклад, редагуємо "index.html"), нам потрібно додати зміни в індекс, щоб вони потрапили в коміт. Для цього ми використовуємо команду:

```
git add index.html
```

Якщо ми хочемо додати кілька змін, треба спочатку проіндексувати їх перед комітом.

Якщо ми змінюємо файл після того, як він вже був доданий в індекс, але ще не був зафіксований в коміт, то при виконанні коміту відбудеться фіксація лише останнього проіндексованого стану файлу. Якщо ми додали файл, але після цього змінили його, треба буде знову виконати команду `git add` для цього файлу перед комітом.

Якщо ми не хочемо зберігати зміни в конкретному файлі, можемо повернути його до стану, що зберігається в репозиторії:

```
git checkout -- index.php
```

Ця команда скине зміни в файлі *index.php* і відновить його до того стану, який є в репозиторії.

1. Коміт змін

Після того як всі файли додані та перевірені, потрібно зафіксувати зміни в репозиторії. Для цього використовується команда:

```
git commit -a -m "Ініціалізація репозиторію"
```

Опція **-a** означає, що всі файли, які були додані або змінені, будуть автоматично поміщені в індекс перед комітом. Ви також вказуєте повідомлення коміту, в якому описуєте зміни.

Опція **-a** означає, що всі файли, які були додані або змінені, будуть автоматично поміщені в індекс перед комітом. Ви також вказуєте повідомлення коміту, в якому описуєте зміни.

3.8 Робота з командою `git log` та перегляд історії комітів

Команда `git log` відображає історію комітів у репозиторії. Вона показує:

- a. Хеш коміту
- b. Ім'я автора
- c. Дату коміту
- d. Повідомлення коміту

У результатах ви побачите всі коміти, починаючи з останнього (згори) і закінчуючи першим (знизу).

Коли журнал комітів занадто довгий, ви можете вийти, натиснувши клавішу **Q**.

У Git є вбудована система допомоги, яка пояснює, як використовувати кожну команду. Для команди `git log` це робиться так:

```
git help log
```

Щоб зробити виведення більш компактним, можна використовувати флаг `--oneline`:

```
git log --oneline
```

Це покаже короткі хеші комітів і їх повідомлення, що зручно для швидкого перегляду. Щоб побачити тільки коміти, зроблені за останні два тижні, використовуйте флаг `--since`:

```
git log --since="2 weeks ago"
```

Використовуючи флаг `-p`, можна побачити зміни, які були внесені в кожному коміті:

```
git log -p
```

Це покаже різницю між станом файлів до і після коміту.

Щоб відобразити останні кілька комітів, можна додати флаг `-n`:

```
git log -n 2
```

Ця команда покаже тільки останні два коміти.

3.9 Робота з гілками в Git

Гілки в Git дозволяють ефективно працювати над різними частинами проєкту паралельно, не порушуючи стабільності основної версії.

1. Створення гілки

a. За замовчуванням є основна гілка **master**.

b. Для розробки нової функції або виправлення багів створюємо нову гілку:

```
git checkout -b new-feature
```

c. Всі зміни фіксуються в новій гілці, не впливаючи на основну.

2. Злиття гілок (merge)

Мерж – це об'єднання змін з однієї гілки в іншу. Якщо ви працюєте в різних гілках, мерж дозволяє зібрати всі зміни в одну гілку.

Після завершення роботи зливаємо гілку з основною:

```
git checkout master  
git merge new-feature
```

Якщо є конфлікти, їх треба вирішити вручну. Завжди перевіряйте, чи правильно працює код після злиття.

3. Робота з критичними помилками (Hotfix)

a. Якщо на продакшен сервері є критичний баг, створюємо окрему гілку для виправлення:

```
git checkout -b hotfix
```

Виправляємо баг і зливаємо зміни з основною гілкою:

```
git checkout master
```

```
git merge hotfix
```

3.10 Основні команди Git:

1. Ініціалізація репозиторію

```
git init
```

Створює новий локальний репозиторій у поточній папці.

2. Клонування репозиторію

```
git clone <url>
```

Копіює віддалений репозиторій на ваш комп'ютер.

3. Перевірка статусу

```
git status
```

Показує стан файлів: які зміни не зафіксовані, не додані або закомічені.

4. Додавання файлів до індексу

```
git add <файл>    # Додає конкретний файл  
git add .          # Додає всі зміни
```

Підготовка файлів для коміту.

5. Фіксація змін (коміт)

```
git commit -m "Опис змін"
```

Зберігає зміни в локальному репозиторії з повідомленням.

6. Перегляд історії змін

```
git log
```

Показує історію комітів.

7. З'єднання з віддаленим репозиторієм

```
git remote add origin <url>
```

Прив'язує локальний репозиторій до віддаленого.

8. Відправка змін у віддалений репозиторій

```
git push origin <branch>
```

Надсилає локальні зміни у віддалений репозиторій.

9. Отримання змін з віддаленого репозиторію

```
git pull origin <branch>
```

Завантажує зміни та об'єднує їх із поточною гілкою.

10. Створення нової гілки

```
git branch <назва>
```

Створює нову гілку.

11. Перехід на іншу гілку

```
git checkout <назва>
```

Перемикає на зазначену гілку.

12. Злиття гілок

```
git merge <назва>
```

Об'єднує зміни з іншої гілки в поточну.

13. Видалення гілки

```
git branch -d <назва>
```

Видаляє локальну гілку.

14. Скасування змін

a. Відмінити додавання файлу: `git reset <файл>`

b. Відмінити останній коміт: `git reset --soft HEAD~1`

Додаткові команди:

- Перегляд змін у файлах: `git diff`

Відновлення файлів: `git checkout -- <файл>`

4. JAVASCRIPT

JavaScript – прототипно-орієнтована скриптова мова програмування. Найширше застосування знаходить в браузерях як мова сценаріїв для надання інтерактивності web-сторінок.

Специфікація (формальний опис синтаксису, базових об'єктів і алгоритмів) мови Javascript називається ECMAScript. Сучасний стандарт – це ECMA-262 5.1. Він підтримується всіма сучасними браузерами, крім Internet Explorer <9. Стандарт для JavaScript не називається просто JavaScript тому, що JavaScript™ - зареєстрована торгова марка, що належить корпорації Oracle. ECMAScript було обрано, щоб зберегти специфікацію незалежної від власників торгової марки.

JavaScript – потужна мова, але на її виконання в браузері накладені деякі обмеження. Це зроблено для безпеки користувачів, щоб зловмисник не міг за допомогою JavaScript отримати особисті дані або якось нашкодити комп'ютеру користувача. У браузері Firefox існує спосіб підписування скриптів з метою обходу частини обмежень, але він нестандартний і не кросс-браузерний.

Унікальність JavaScript полягає в:

1. Повній інтеграції з HTML/CSS.
2. Підтримці всіма браузерами по замовчуванню.
3. Простоті: прості речі робляться просто.

4.1. Способи додавання скриптів на сторінку

Скрипт – це послідовність команд певному виконавцю, що їх виконує. JS скрипти – це послідовність операторів, що вказують браузеру як відображати вебсторінку.

Скрипт можна додати в будь-яке місце HTML документу, за допомогою елемента `script`. Стандарти HTML4 вимагають обов'язкового вказування атрибуту `type= "text/javascript"`. Є кілька способів для додавання скрипту на веб сторінку:

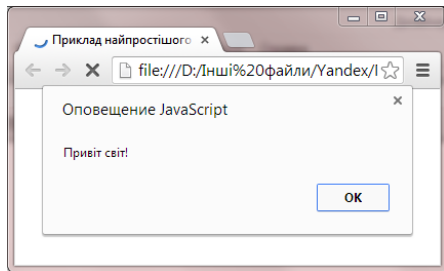
1. Безпосереднє включення в текст сторінки. Для цього оператори записують безпосередньо між тегами

```
<script type="text/javascript">  
    скрипт  
</script>
```

Приклад найпростішого скрипта, що виводить привітання.

Для його запуску необхідно створити html сторінку, що містить наступний текст і відкрити за допомогою браузера.

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Приклад найпростішого скрипта</title>  
    <script type="text/javascript">  
      alert("Привіт світ!")  
    </script>  
  </head>  
<body>  
  Скрипт виконано.  
</body>  
</html>
```



2. Також скрипти можна зберігати в окремих файлах. Для цього необхідно створити текстовий файл, що містить скрипт і зберегти його з розширенням js. А на сторінку він підключається за допомогою елемента `script` з атрибутом `src`.

```
<script type="text/javascript" src="шлях/ім'я.js"> </script>
```

Якщо файл, що містить скрипт називається "Hello world" і міститься в тій же папці, що і html сторінка, то її текст буде наступним:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Приклад найпростішого скрипта</title>
    <script type="text/javascript" src="Hello
world.js"> </script>
  </head>
  <body>
    Скрипт виконано.
  </body>
</html>
```

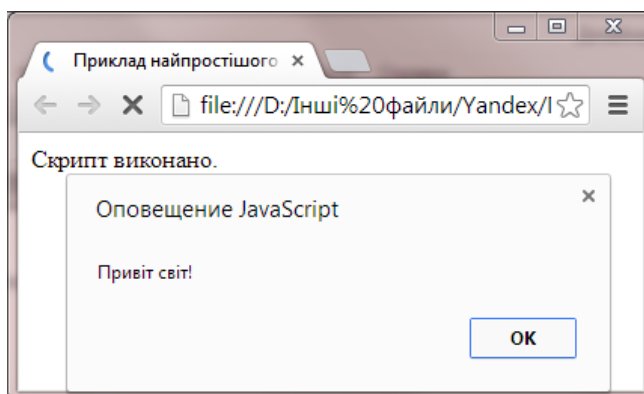
А файл скрипта буде містити єдиний рядок:

```
alert("Привіт світ!");
```

Надалі будемо вважати, що html сторінка вже створена і ми редагуємо тільки відповідний файл зі скриптом.

Варто зазначити, що інтерпритатор будує сторінку послідовно, відповідно знайшовши на ній скрипт – відкладає подальшу побудову до закінчення виконання скрипта. Якщо скрипт розмістити в заголовочній частині документа, то він виконається до створення тіла документа, відповідно спочатку з'явиться повідомлення на фоні порожнього документа і тільки після його закриття з'явиться надпис "Скрипт виконано". Якщо ж розмістити елемент скрипта в тілі документа після відповідного надпису, то вікно з'явиться на фоні документа, що вже містить надпис.

```
<body>  
    Скрипт виконано.  
    <script type="text/javascript" src="../../Інші файли/Hello  
world.js"></script>  
</body>
```



Якщо документ містить кілька скриптів, то вони виконуються послідовно, в порядку розміщення на сторінці.

Оператори розділяються за допомогою символу ";". Перехід на новий рядок не розділяє оператори, але при виконанні скрипта інтерпритатор намагається виправити помилки і може сам додати

розділення, але це не завжди приводить до правильного результату.

Коментарі.

Однорядкові коментарі починаються з подвійного слеша `//`.

Багаторядкові коментарі починаються слешем-зірочкою `/*` і закінчуються зірочкою-слешем `*/`.

4.2. Опис змінних та констант

Змінна – це певні данні збережені у пам'яті під певним іменем. Для використання змінних їх слід оголосити. Оголошення здійснюється за допомогою оператора `var`.

```
var змінна;
```

При оголошенні також можна задавати початкове значення змінної.

```
var змінна = початкове значення;
```

Ім'я кожної змінної повинно бути унікальним правильним ідентифікатором. Правильний ідентифікатор на мові JS повинен містити тільки букви латинського алфавіту, цифри та символи `"$"` і `"_"`. При цьому ідентифікатор не може починатись з цифри і співпадати із зарезервованими словами.

Правильні ідентифікатори: `myID`, `identifier`, `x`, `$`, `abc123`, `my_id`, `_true`.

Неправильні: `my id`, `Ok!`, `12th`, `100`, `for`.

Мова JS є регістро чутливою (розрізняє ВЕЛИКІ та малі букви). Тому `myid` та `MyID` – це різні ідентифікатори.

Змінні оголошуються глобально: вони доступні в будь-якому скрипті, що є на сторінці.

Змінні в JS не є типізованими: не можна строго вказати тип змінної, вона може змінювати свій тип, залежно від значення.

Приклад:

```
var x, y = 2; // x - змінна, y - змінна цілого типу, рівна 2
x = y / 8;    // x - дійсного типу, рівна 0,25
x = "String"; // змінює свій тип на рядковий.
```

Правильний вибір імені змінної.

Правило 1: Ніякого трансліта. Тільки англійська мова.

Неприйнятні:

- var moiTovari;
- var cena;
- var ssilka;

Підійдуть:

- var myGoods;
- var price;
- var link;

Чим поганий трансліт? По-перше, серед розробників усього світу прийнято використовувати англійську мову для імен змінних. І, якщо ваш код потім потрапить до когось іншого, наприклад, ви будете в команді більше ніж з однієї людини, то великий шанс, що трансліт йому не сподобається.

Правило 2: Використовувати короткі імена тільки для змінних «місцевого значення».

Називати змінні іменами, що не несуть змістовного навантаження, наприклад `a`, `e`, `p`, `mg` – можна тільки в тому випадку, якщо вони використовуються в невеликому фрагменті коду та їх застосування очевидно. Назва змінної повинна бути зрозумілою. Іноді для цього потрібно використовувати кілька слів.

Правило 3: Змінні з декількох слів пишуться разом.

Наприклад:

`var borderLeftWidth;`

Цей спосіб запису називається «**верблюжий регістр**» або, по-англійськи, «`camelCase`».

Існує альтернативний стандарт, коли кілька слів пишуться через знак підкреслення `'_'`:

`var border_left_width;`

Переважно в JavaScript використовується варіант `borderLeftWidth`, зокрема у вбудованих мовних та браузерних функціях. Тому доцільно зупинитися на ньому.

Правило 4: Ім'я змінної має максимально чітко відповідати збереженням в ній даним.

`borderLeftWidth` : `leftBorderWidth`.

Приклад:

Оголосіть дві змінні: `admin` і `name`. Запишіть в `name` рядок «Василь». Скопіюйте значення з `name` в `admin`. Виведіть `admin` (повинно вивести «Василь»).

```
var admin, name; // Дві змінних через кому
name = "Василь";
admin = name;
```

```
alert (admin); // "Василь"
```

Константи описуються за допомогою оператора `const`.

```
const КОНСТАНТА = значення;
```

Константи, на відміну від змінних, не можуть змінювати свого значення. Традиційно імена констант записують великими літерами.

Приклад: `const PI = 3.14, ZERO = 0, NAME = "Name";`

Константа – це змінна, яка ніколи не змінюється. Як правило, їх називають великими літерами, через підкреслення, наприклад:

```
var COLOR_RED = "# F00";  
var COLOR_GREEN = "# 0F0";  
var COLOR_BLUE = "# 00F";  
var COLOR_ORANGE = "# FF7F00";  
var color = COLOR_ORANGE;  
alert (color); // # FF7F00
```

Технічно, константа є звичайною змінною, тобто її можна змінити. Але ми домовляємося цього не робити.

4.3. Типи даних

Число «number»

```
var n = 123;  
n = 12.345;
```

В JavaScript можна записувати числа не тільки в десятковій, але і в шістнадцятковій (запис починається з `0x`), а також вісімковій (починається з `0`) системах числення: `0xFF` це 255 в шістнадцятковій системі, а `010` – 8 у вісімковій системі. Також доступний запис «з плаваючою крапкою», який виглядає як

мантиса є ступінь. Наприклад: $1e3$ – це $1 \cdot 10^3$, тобто 1000, $3e-5$ $= 3 \cdot 10^{-5} = 0.00003$.

В пам'яті числа зберігаються в форматі IEEE-754, також відомому як **«double precision»**. В ньому на число виділяється рівно 8 байт (64 біти). Деякі дробові числа в десятковому форматі мають скінченний запис, а в двійковій системі числення це нескінченний дріб. Так нескінченним дробом є $0.1_{10}=0,(00011)_2$. Двійкове значення нескінченних дробів зберігається лише до певного знака, тому виникає неточність.

```
alert(0.1 + 0.2); // 0.30000000000000004, а не 0.3 як  
очікувалось
```

Втрата точності відбувається і з дуже великими числами:

```
alert(999999999999999999999999); //1e+22
```

Числа, як і всі інші типи в JavaScript є об'єктами. Кожне число має метод `toFixed` (точність), який округлює число до заданої точності і повертає результат у вигляді рядка

```
var n = 12.345;
alert (n.toFixed (1)); // "12.3"
```

Викликати його можна не тільки у змінних числового типу, а й у числових літералах.

```
alert ((12.345).toFixed (1)); // "12.3"
```

Кожне число має вбудований метод `toFixed(кількість знаків)`, який округлює число до заданої точності і повертає результат у вигляді рядка. Кількість цифр після десяткового знака повинна знаходитися в діапазоні від 0 до 20 включно. Якщо число більше $1e+21$, то метод поверне просто рядок в експоненціальному записі.

Основні оператори

Над числами можна виконувати звичайні арифметичні, та деякі спеціальні операції.

<code>x+y</code>	Сума
<code>x-y</code>	Різниця
<code>x*y</code>	Добуток
<code>x/y</code>	Частка
<code>x%y</code>	Остача від ділення
<code>-x</code>	Зміна знаку на протилежний
<code>+x</code>	Збереження знаку.
<code>x++, ++x</code>	Інкремент: збільшення на 1
<code>x--, --xx</code>	Декремент: зменшення на 1
<code>. [] ()</code>	Доступ до полів, індексація масивів, виклики функцій та групування виразів
<code>++ -- - ~ ! delete new typeof void</code>	Унарні оператори, тип даних, що повертаються, створення об'єктів, невизначені значення
<code>* / %</code>	Множення, ділення, множення по модулю
<code>+ - +</code>	Додавання, віднімання, об'єднання рядків
<code><< >> >>></code>	Зсув бітів
<code>< <= > >= instanceof</code>	Менше, менше чи рівне, більше, більше чи рівне, instanceof
<code>== != === !==</code>	Рівність, нерівність, строга рівність, строга нерівність
<code>&</code>	Побітове І
<code>^</code>	Побітове виключне АБО
<code> </code>	Побітове АБО
<code>&&</code>	Логічне І
<code> </code>	Логічне АБО

?:	Умовний оператор
= OP=	Присвоєння, присвоєння з операцією (наприклад += та &=)
,	Обчислення кількох виразів

Операнд – те, до чого застосовується оператор. Наприклад: 5 * 2 – оператор множення з лівим і правим операндами. Інша назва: «аргумент оператора».

Унарним називається оператор, який застосовується до одного виразу. Наприклад, оператор унарний мінус "-" змінює знак числа на протилежний:

```
var x = 1;
x = -x;
alert(x); // -1, унарний мінус
```

Бінарним називається оператор, який застосовується до двох виразів. Той же мінус існує і в бінарній формі:

```
var x = 1, y = 3;
alert( y - x ); // 2, бінарний мінус
```

Зазвичай за допомогою плюса "+" додають числа. А ле якщо бінарний оператор "+" застосувати до рядків, то він їх об'єднує в один:

```
var a = "мій" + "рядок";
alert( a ); // мійрядок
```

Якщо хоча б один аргумент є рядком, то другий буде також перетворений на рядок! Унарний плюс перетворює рядок в число.

Інкремент та декремент можна застосувати тільки до змінної. Код 5++ дасть помилку. Різниця між префіксною та постфіксною формами виявляється там, де потрібно не тільки збільшити/зменшити змінну, але й отримати результат. При

використанні постфіксної форми спочатку відбуваються обрахунки, а потім змінюється значення змінної. З префіксною формою навпаки: спочатку змінюється значення, а потім відбуваються обрахунки. Нехай $x=1$, тоді після виконання $y=3*(x++)$ буде $y=3$, $x=2$. Після $y=3*(++x)$ отримаємо $y=6$, $x=2$.

Кома дозволяє перераховувати вирази, розділяючи їх комою ','. Кома – єдиний оператор, пріоритет якого нижче присвоювання. У виразі $a = (5,6)$ для явного задання пріоритету використані дужки, інакше оператор '=' виконався б до коми ',', вийшло б $(a = 5), 6$.

Зазвичай він використовується в складі більш складних конструкцій, щоб зробити кілька дій в одному рядку. Наприклад

```
for (a = 1, b = 3, c = a*b; a < 10; a++) {  
    ...  
}
```

Існують спеціальні числові значення Infinity (нескінченність) і NaN (помилка обчислень).

Infinity – особливе числове значення, яке поводить себе як математична нескінченність ∞ . Infinity більше ніж будь-яке число. В результаті додавання/віднімання/множення/ділення нескінченності на число отримуємо нескінченність. Відношення числа до нескінченності дає 0. Також існує мінус нескінченність -Infinity.

```
alert (1/0); // Infinity
```

Якщо математична операція не може бути здійснена, то повертається спеціальне значення NaN (Not-A-Number, не

число). Так воно отримується в результаті $0/0$, ∞/∞ , $\infty-\infty$, $\sin(\infty)$, $\sqrt{-1}$ та ін. Значення NaN – єдине, у своєму роді, яке не дорівнює нічому, включаючи себе. В результаті будь-якої операції з NaN отримується NaN. Значення NaN можна перевірити спеціальною функцією `isNaN(n)`, яка повертає true якщо аргумент – NaN і false для будь-якого іншого значення. Функція `isFinite(n)` повертає true тільки тоді, коли n звичайне число, а false якщо n одне зі спеціальних значень: NaN, Infinity чи -Infinity. Якщо аргумент не число, то він буде автоматично перетворений на число.

```
alert ("нечисло" * 2); // NaN, помилка
```

При строгому перетворенні, якщо рядок не є точно числом, то отримаємо NaN. Єдиний виняток – пробіли на початку та в кінці рядка, які ігноруються. А також порожній рядок перетворюється на 0.

Якщо один з операндів арифметичної операції, крім додавання, є числом тоді інший операнд буде також перетворений на число. Унарні арифметичні операції також перетворюють аргумент в число, тому найпростішим способом перетворення на число – унарна операція збереження знаку.

У web-програмуванні багато значення не є точно числами, але мають числовий зміст. Наприклад, метрики CSS: 10pt, -1cm чи 12%. Для їх зручного перетворення є функції м'якого перетворення. Вони перетворюють рядок символ за символом,

поки це можливо. Звичайно, існують ситуації, коли вони повертають NaN. Це відбувається при помилці в першому ж символі. **parseInt** перетворює рядок в ціле число, а **parseFloat** – дробове. **parseInt** може мати і другий аргумент, що вказує систему числення з якої відбувається перетворення. Це робити не обов’язково, якщо запис починається з 0х – явної вказівки на шістнадцяткову систему числення.

```
parseInt ('12px ')          // 12
parseFloat ('12 .3.4')      // 12.3
parseInt ('a123')           // NaN
parseInt ('0xFF')           // 255
```

В JavaScript немає вбудованих математичних констант та функцій, але вони реалізовані як властивості та методи простору імен Math.

Основні методи:

Math.floor (x)	Округлення вниз
Math.ceil (x)	Округлення вгору
Math.round (x)	Округлення до найближчого цілого
Math.acos (x)	Арккосинус x (в радіанах)
Math.asin (x)	Арксинус x (в радіанах)
Math.atan (x)	Арктангенс x (в радіанах)
Math.atan2 (y, x)	Арктангенс кута до точки (y, x) Виклик еквівалентний Math.atan (y/x)
Math.sin (x)	Синус x (в радіанах)
Math.cos (x)	Косинус x (в радіанах)
Math.tan (x)	Тангенс x (в радіанах)
Math.sqrt (x)	Квадратний корінь з x

Math.log (x)	Натуральний логарифм x
Math.pow (x, y)	Піднесення до степеня, повертає x^y , наприклад Math.pow (2,3) = 8
Math.abs (x)	Абсолютне значення числа
Math.exp (x)	Експонента e^x
Math.max (a, b, c ...)	Найбільший зі списку аргументів
Math.min (a, b, c ...)	Найменший зі списку аргументів
Math.random (x)	Псевдо-випадкове число в інтервалі [0,1) – тобто між 0 (включно) та 1 (не включаючи). Генератор випадкових чисел ініціалізується поточним часом
Math.PI	Число π
Math.E	Константа Ейлера, e

Кожне число має вбудований метод **toFixed(кількість знаків)**). Повертає рядкове представлення числа без використання експоненціального запису, з вказаною кількістю цифр після коми. Кількість цифр після десяткового знака повинна знаходитися в діапазоні від 0 до 20 включно. Якщо не вказано, то вважається рівним 0. Число округлюється при необхідності, і дробова частина доповнюється нулями до потрібної довжини. Якщо число більше $1e+21$, то метод просто рядок в експоненціальному записі.

Приклад. Скласти скрипт для генерування випадкового числа з проміжку [a,b].

```
var a=prompt('a=', ''), b=prompt('b=', '');
var rnd=a+Math.floor((b-a+1)*Math.random());
alert(rnd);
```

Рядок «string»

```
var str = "студенти вчили javascript";
```

```
str = 'Одинарні лапки теж підійдуть';
```

В JavaScript одинарні та подвійні лапки рівноправні. Можна використовувати або ті, або інші.

Тип символ не існує, є лише рядок. У деяких мовах програмування є спеціальний тип даних для одного символу. Наприклад, у мові C це char. В JavaScript є тільки тип «рядок» string. Що, треба сказати, цілком зручно.

Булевий (логічний) тип «boolean»

Має всього два значення – true (істина) і false (неправда). Використовується для зберігання значення типу так/ні. Отримується в результаті виконання операторів порівняння та в умовах операторів if, while.

Більше	a > b
Менше	a < b
Більше або дорівнює	a >= b
Менше або дорівнює	a <= b
Рівне. a=b це присвоювання!	a == b
Не рівно, ≠	!=
Тотожно рівно	a === b
Не тотожно рівно	a !== b

При порівнянні значення перетворюються до числа, крім випадку, коли обидва значення – рядки.

Звичайна рівність не може відрізнити 0 від false. Для перевірки рівності без перетворення типів використовуються оператори тотожної рівності === і !== . Вони порівнюють без приведення типів. Якщо типи різні, то такі значення завжди нерівні.

Спеціальне значення «null»

null – спеціальне значення, яке утворює свій окремий тип. Воно має зміст «нічого». Значення null, що складається з єдиного значення. В JavaScript null – це просто спеціальне значення, яке має сенс «**нічого**» або «**значення невідоме**».

```
var age = null;
```

Спеціальне значення «undefined»

undefined – спеціальне значення, яке утворює свій власний тип. Воно має зміст «**значення не присвоєно**». Якщо змінна оголошена, але в неї нічого не записано, то її значення як раз і є **undefined**.

```
var x;
```

```
alert (x); // Виведе "undefined"
```

Можна привласнити undefined і в явному вигляді, хоча це робиться рідко:

```
var x = 123;
```

```
x = undefined;
```

```
alert (x); // "Undefined"
```

Значення **null** і **undefined** дорівнюють == один одному і не рівні чому б то не було ще. Це буквально написано в специфікації мови. При перетворенні в число **null** стає 0, а **undefined** стає NaN (числове «значення-помилка»). Це приводить до непередбачуваних наслідків: null не більше і не рівне нулю, але більше рівне за нього. А значення **undefined** взагалі не можна порівнювати.

```
alert (null > 0); // false
alert (null > 0); // false
alert (null >= 0); // true
alert (undefined < 0); // false
alert (undefined > 0); // false
alert (undefined == 0); // false
```

У явному вигляді `undefined` зазвичай не присвоюють, оскільки це суперечить його змісту. Для запису в змінну «порожнього» або «невідомого» значення використовується `null`.

Об'єкти «object»

Оголошуються об'єкти за допомогою фігурних дужок `{...}`, наприклад:

```
var user = {name: "Вася"};
```

4.4. Логічні оператори

В JavaScript підтримуються оператори `||(АБО)`, `&&(І)` та `!(НЕ)`.

Вони називаються «логічними», але в JavaScript можуть застосовуватися до значень будь-якого типу і повертають також значення будь-якого типу.

Логічне `||(АБО)` в класичному програмуванні працює таким чином: «якщо хоча б один з аргументів `true`, то повертає `true`, інакше – `false`».

```
alert( true || true ); // true
alert( false || true ); // true
alert( true || false ); // true
alert( false || false ); // false
```

JavaScript обчислює кілька АБО зліва направо. При цьому, щоб економити ресурси, використовується так званий «короткий цикл обчислення».

Припустимо, обчислюються кілька АБО поспіль: `a || b || c || ...`. Якщо перший аргумент – `true`, то результат свідомо буде `true` (хоча б одне із значень – `true`), й інші значення ігноруються.

При запуску прикладу нижче привласнення `x` не відбудеться:

```
var x;  
true || (x = 1); // просто обчислимо АБО, без if  
alert (x); // undefined, x не присвоєно
```

А в прикладі нижче перший аргумент – `false`, так що АБО спробує обчислити другий, запустивши тим самим присвоєвання:

```
var x;  
false || (x = 1);  
alert (x); // 1
```

Оператор АБО обчислює рівно стільки значень, стільки необхідно – до першого `true`. Він повертає те значення, на якому зупинилися обчислення.

У класичному програмуванні **&&(І)** повертає `true`, якщо обидва аргументи істинні, а інакше – `false`.

```
alert(true && true); // true
```

```
alert(false && true); // false
```

```
alert(true && false); // false
```

```
alert(false && false); // false
```

До **І** застосуємо той же принцип «короткого циклу обчислень», але трохи по-іншому, ніж до АБО. Якщо лівий аргумент – `false`,

оператор `!` повертає його і закінчує обчислення. Інакше – обчислює і повертає правий аргумент.

Оператор **!(НЕ)** спочатку приводить аргумент до логічного типу `true` / `false`. Потім повертає протилежне значення.

```
alert(!true); // false
alert(!0); // true
```

4.5. Ввід та вивід даних. Взаємодія з користувачем

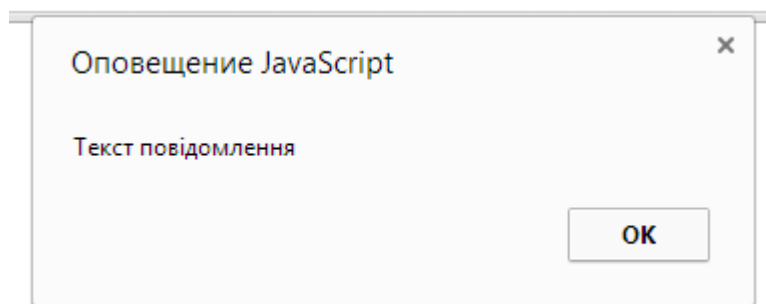
Для обміну даними з користувачем є різні способи: за допомогою модальних вікон, консолі, форм, вмісту сторінки та ін.

Розглянемо перший спосіб.

Для виведення використовується підпрограма `alert`:

`alert (текст повідомлення);`

В результаті виводиться модальне вікно із повідомленням та єдиною кнопкою `Ok`.

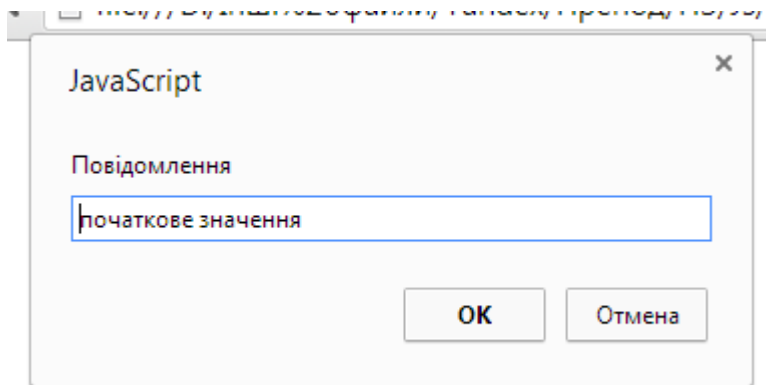


Єдиний параметр приводиться до текстового типу.

Для введення даних будемо користуватись функцією `prompt`;

`prompt (повідомлення, початкове значення);`

В результаті виводиться модальне вікно, яке містить інформаційне повідомлення, поле для вводу з початковим значенням та кнопки Ok та Відміна. Зазвичай, як другий параметр, використовується порожній рядок. Якщо його не вказати, то в деяких браузерах значення буде undefined.



Функція повертає значення, що було введено користувачем у відповідне поле. Значення повертається як текстова величина. Якщо необхідно ввести числове значення то необхідно скористатись функціями переведення `parseInt`, `parseFloat` чи оператором збереження знаку `+`. В разі натискання кнопки відміни функція повертає `null`.

Приклад:

```
var someString = prompt ('Введіть деякий рядок','');  
var intNumber = parseInt(prompt ('Введіть ціле число',''));  
var x = +prompt ('Введіть число','');
```

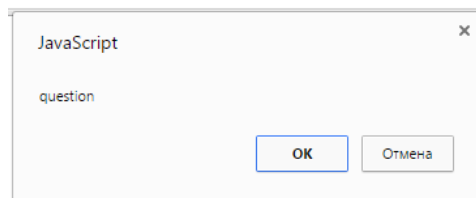
Браузер може заборонити виведення модальних вікон.

confirm

```
result = confirm('question');
```

confirm виводить вікно з повідомленням і двома кнопками: ОК і Відміна.

Результатом буде **true** при натисненні ОК і **false** – при Відміна (ESC).



Alert виводить повідомлення.

Prompt виводить повідомлення і чекає, поки користувач введе текст, а потім повертає введенне значення або null, якщо введення скасований (CANCEL / Esc).

Confirm виводить повідомлення і чекає, поки користувач натисне «ОК» або «CANCEL» і повертає true / false.

4.6. Умовні оператори

Іноді, залежно від певних умов, потрібно виконати різні дії. Для цього використовується умовний оператор. Він може бути в повній

if (умова){оператори, якщо умова виконується}else{оператори, якщо умова не виконується};

або скороченій формі

if (умова){оператори, якщо умова виконується}.

Приклад. Скласти скрипт для запиту від користувача віку. Якщо його вік <18 років то повідомити, що голосувати йому ще рано.

```
var age=parseInt(prompt('Ваш вік',''));  
if (age<18) {alert('Доступ заборонений')} else {alert('Ласкаво  
просимо')};
```

Іноді, залежно від умови, потрібно присвоїти змінній певне значення.

Змінна = умова ? значення, якщо виконується умова : значення,
якщо невиконується умова;

Приклад. Скласти скрипт для знаходження більшого з двох чисел.

```
var a=parseFloat(prompt('Введіть перше число',''));  
var b=parseFloat(prompt('Введіть друге число',''));  
var max=(a>b)?a:b;  
alert('Більше число рівне '+max);
```

Конструкція switch замінює собою відразу кілька if. Це більш наочний спосіб порівняти вираз (мається на увазі оператор «===») відразу з декількома варіантами.

```
switch(селектор) {  
    case значення1 : оператори; break; // if (селектор ===  
значення1)  
    case значення2: оператори; break; // if (селектор  
=== значення2)  
    ...  
    case значенняn: оператори; break; // if (селектор ===  
значенняn)
```

default: оператори //інакше

};

Приклад. Скласти скрипт, який запитує від користувача його браузер і виводить його іконку в тілі документа.

```
var brauser=prompt('Ваш браузер','');
var x='';
switch (brauser.toUpperCase()){
  case 'FIREFOX':
    x='ff';
    break;
  case 'CHROME':
    x='c';
    break;
  case 'OPERA':
    var version=prompt('Версія','');
    version=version.toUpperCase();
    if ((version=='MOBILE')||(version=='MINI'))
      {alert('Мобільний браузер')}
      else{x='o'};
    break;
  case 'INTERNET EXPLORER':
    x='ie';
    break;
  default:
    alert('Невідомий браузер');
}
if (x!=''){
  document.writeln('<h1>'+brauser+'</h1>');
  document.writeln('');
}
```

4.7. Цикли

Для повторення виконання однотипних операцій призначені цикли. В JavaScript, як і в більшості інших мов, є три типи циклів: з передумовою, післяумовою та параметром.

Цикл з передумовою має таку структуру:

```
while (умова) {  
    тіло циклу  
};
```

Поки умова вірна – виконується код з тіла циклу. Якщо умова виконується завжди, то цикл повинен виконуватися вічно (в теорії). На практиці, браузер виведе повідомлення про скрипт, який «завис» і користувач його зупинить.

Приклад. Скласти скрипт зворотного відліку, який виводить числа від 5 до 1, а потім повідомлення «БУМ!»

```
var i=5;  
while (i>0){  
    alert(i--);  
}  
alert('БУМ!');
```

Цикл з післяумовою має таку структуру:

```
do {  
    тіло циклу  
} while (умова);
```

Особливість даного циклу – тіло буде виконане хоча б один раз, незалежно від умов. Як і в циклі з передумовою, тіло виконується поки умова вірна.

Приклад. Скрипт зворотного відліку:

```
var i=5;
do {
    alert(i--);
}while (i>0);
alert('Кінець!');
```

Приклад. Скласти скрипт, який просить від користувача обчислити $a+b$, де a і b – цілі числа, що генеруються випадковим чином перед кожною спробою.

```
do {
    var a=Math.floor(Math.random()*100),
    b+=(Math.random()*100).toFixed(0);
    var res+=prompt(a+''+b+'=', '');
} while ((a+b!= res));
alert('Вірно');
```

Найчастіше застосовується **цикл з параметром**

for (початкове значення; умова; зміна параметру) {
 тіло циклу
}

Будь-яка частина for може бути пропущена. Наприклад, можна прибрати початкове значення. Можна прибрати і крок:

```
for (;умова;) { //цикл перетворився на аналог while (умова)
}
```

А можна і взагалі прибрати все, отримавши нескінченний цикл:

```
for (;;) // Буде виконуватися вічно
```

При цьому самі крапки з комою «;» обов'язково повинні бути присутніми, інакше буде помилка синтаксису.

Приклад. Скрипт зворотного відліку:

```
for (var i=5;i>0;i--){
    alert(i);
}
alert(' Кінець!');
```

Приклад. Користувач вводить числа поки не натисне відміну.

Знайти їх середнє арифметичне.

// з передумовою

```
var s=0,k=0, x=prompt('Введіть число','');
while (x!=null){
    s=s+(+x);
    k++;
    x=prompt('Введіть число','');
}
// частина ідентична для всіх циклів
if (k!=0) {
    alert(s/k) }
else{
    alert('не введено жодного числа')
}
```

// з післяумовою

```
var s=0,k=0;
do {
    var x=prompt('Введіть число','');
    if (x!=null) {
        s+=(+x);
        k++;};
} while (x!=null);

// з параметром
for (var s=0,k=0,x=prompt('Введіть число','');x!=null;k++){
s=s+(+x);
    x=prompt('Введіть число','');
}
```

Вийти з циклу можна не тільки при перевірці умови але й, взагалі, в будь-який момент. Цю можливість забезпечує оператор `break`.

Модифікуємо скрипт так, щоб при натисканні відміни в користувача перепитувало чи він справді хоче вийти.

Приклад:

```
do {
    var a=Math.floor(Math.random()*100),
    b+=(Math.random()*100).toFixed(0);
    var res=prompt(a+'+'+b+'=', '');
    if (res==null){
        if (confirm('Ви справді бажаєте вийти?')){
            break;
        }
    }
} while ((a+b!= +res));
if (res!=null){ alert('Вірно')};
```

Буває, потрібно вийти одночасно з декількох рівнів циклу. Звичний виклик `break` не може перервати два цикли відразу. Для цього існують мітки.

Мітка має вигляд **ім'я:**. Ім'я повинно бути унікальним. Вона кладеться перед циклом.

Приклад:

```
exit:
for (var i=0;i<2;i++) {
    for (var j=0;j<2;i++){
        var input=prompt('a['+i+', '+j+']', '');
        if (input==null) break exit; // (*)
    }
};
alert('Готово!');
```

Директива `continue` припиняє виконання поточної ітерації циклу та виконує перехід до наступної. Вона також може застосовуватись з мітками.

Приклад. Вивеси усі прості числа менші від `n`.

```
var n=prompt('n=','');
document.writeln('<h1>Прості числа до '+n+'</h1><br>');
next: for (var i=1;i<=n;i++){
    for (var j=2;j<=Math.sqrt(i);j++){
        if (i%j==0){continue next}; //Число не просте,
        перевіряємо наступне.
    };
    document.writeln(i);
};
```

4.8. Функції

Часто необхідно повторювати одну і ту ж дію багато разів. Наприклад, красиво вивести повідомлення привітання, при вході відвідувача на сайт. Щоб не повторювати один і той же код в багатьох місцях, існують підпрограми. В JS всі підпрограми є функціями. Існують вбудовані функції, з якими ви вже зустрічались – це `alert`, `prompt`, `confirm` та ін. А можна описати і свої.

Синтаксис оголошення функції:

```
function назва(параметр1, параметр2, ... , параметрN) {
    тіло функції;
    return результат;
}
```

Спочатку йде ключове слово `function`, після нього ім'я функції, потім список параметрів в дужках (він може бути порожній) і тіло функції – послідовність команд, що виконується при її виклику. Оголошена функція доступна по імені. Головна мета створення функцій – уникання дублювання коду. Ще одна перевага використання функцій полягає в тому, що якщо знадобиться внести зміни – достатньо змінити код в одному місці: у функції, яка його виконує.

Функція може містити локальні змінні, оголошені через `var`. Такі змінні доступні тільки всередині функції. Блоки `if`, `else`, `switch`, `for`, `while`, `do ... while` та інші не впливають на область видимості змінних. При оголошенні змінної в таких блоках, вона все одно буде видима у всій функції. Неважливо, де саме в функції і скільки разів оголошується змінна. Будь-яке оголошення спрацьовує один раз і поширюється на всю функцію. Функція може звернутися до зовнішньої змінної. При цьому, змінна буде доступна й зовні функції. Якщо всередині функції є оголошена своя локальна змінна з тим же іменем, то всі звернення використовують її і зовнішня змінна залишиться незмінною. Змінні, оголошені на рівні всього скрипта, називають глобальними змінними. Робити глобальними слід тільки ті змінні, які дійсно мають загальне значення для проекту.

Function Declaration

Синтаксис оголошення функції:

`function назва(параметр1, параметр2, ... , параметрN) {`

```
тіло функції;  
return результат;  
}
```

Function Expression (функціональний вираз)

```
var f = function(параметри) {  
  // тіло функції  
};
```

Function Declaration – функція, оголошена в основному потоці коду.

Function Expression – оголошення функції в контексті якого-небудь виразу, наприклад, присвоювання.

Приклад:

```
// Function Declaration  
function sum(a, b) {  
  return a + b;  
}  
alert( sum(5,4) );
```

Приклад:

```
// Function Expression  
var sum = function(a, b) {  
  return a + b;  
}  
alert( sum(5,7) );
```

Основна відмінність між ними: функції, оголошені як Function Declaration, створюються інтерпретатором до виконання коду. Function Declaration можна викликати до оголошення.

new Function

new Function (params, code):

params – параметри функції через кому у вигляді рядка;

code – код функції у вигляді рядка.

Приклад:

```
var sum = new Function('a,b', ' return a+b; ');  
var result = sum(1, 2);  
alert( result ); // 3
```

Named Function Expression

var f = function sayHi(...) { /* тіло функції */ };

Ім'я функціонального виразу (sayHi) доступне тільки зсередини самої функції.

Приклад:

```
var f = function factorial(n) {  
    return n ? n*factorial(n-1) : 1;  
};  
var g = f; //  
alert( g(5) ); // 120
```

4.9. Масиви

На відміну від деяких мов програмування, масиви в JavaScript є складними, впорядкованими неоднорідними типами даних. Тобто в масиві можна зберігати елементи різних типів. Кожен елемент характеризується своїм індексом – унікальним номером.

Синтаксис для створення нового масиву – квадратні дужки зі списком елементів у середині.

var arr = []; //Порожній масив

**var fruits = ["Яблуко", "Апельсин", "Слива"]; // Масив
фруктів з трьома елементами**

var arr = [1, 'Ім\''я', [1,2,3], true]; // різнотипні елементи

Враховуючи, що масив є об'єктом, його можна створити за допомогою конструктора `new Array()`. Якщо як параметр передати одне ціле число, то буде створено порожній масив відповідної довжини, в решті випадків буде створено масив, що складається з аргументів конструктора. Виклик конструктора із від'ємним аргументом призводить до помилки.

var arr = new Array(2,3); // еквівалентно arr = [2, 3]

arr = new Array(3); // arr = [,]

arr = new Array(-3); // Помилка

Через `alert` можна вивести масив цілком. При цьому значення його елементів будуть перераховані через кому:

alert (fruits); // Яблуко, Апельсин, Слива

Загальне число елементів, збережених в масиві, міститься в його властивості `length`:

alert (fruits.length); //3

Зазвичай нам не потрібно самостійно міняти `length`. При зменшенні `length` масив вкорочується. Причому цей процес необоротний, тобто якщо потім повернути `length` назад – значення не відновлюються. Найпростіший спосіб очистити масив – присвоїти властивості `length` нульове значення.

Елементи нумеруються, починаючи з нуля. Щоб отримати потрібний елемент з масиву, вказується його індекс у квадратних дужках:

```
alert (fruits [0]); // Яблуко
```

```
alert (fruits [2]); // Слива
```

Елемент можна завжди замінити:

```
fruits [2] = 'Груша'; // тепер ["Яблуко", "Апельсин",  
"Груша"]
```

Або додати:

```
fruits [3] = 'Лимон'; // тепер ["Яблуко", "Апельсин",  
"Груша", "Лимон"]
```

Слід врахувати, що додавання елемента в позицію, яка перевищує поточну розмірність масиву, призводить до збільшення розмірності. При цьому елементи, значення яких не вказано явно, не займають місце в пам'яті. При виводі масиву через alert на їх позиції нічого не відобразиться, але відповідні коми виводяться. При звертанні безпосередньо до них, значення вважається рівне undefined.

```
var arr[1,2,3];
```

```
arr[5]=5; // в результаті отримаємо масив з 6 елементів  
[1,2,3,,,5]
```

```
alert(a[4]); // undefined
```

```
alert(a.length); //6
```

Можна присвоювати цілі масиви, але необхідно врахувати, що при цьому відбувається копіювання вказівника на масив. Це може

викликати непередбачувані наслідки, тому доводиться копіювати масиви поелементно:

```
var a=[1,2,3],b=a;  
b[1]='Ой!';  
alert(a[1]); // Ой!, а не як очікувалось 2
```

```
var a=[1,2,3],b=[];  
for (var i=0;i<a.length;i++){b[i]=a[i]};  
b[1]='Ой!';  
alert(a[1]); //2 як і має бути
```

Масиви в JavaScript можуть містити в якості елементів інші масиви. Цим можна скористатись для створення багатовимірних масивів, наприклад матриць. Звертатись до елемента необхідно беручи кожен індекс в квадратні дужки.

```
var matrix = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
];  
alert (matrix [1][2]); // елемент в другому рядку, третьому  
стовпцю рівний 6
```

Якщо матрицю неможливо задати явно, то щоб інтерпретатор зрозумів з чим має справу, перед її подальшою обробкою необхідно провести ініціалізацію: спочатку описати матрицю як масив, а потім кожному елементу присвоїти порожній масив.

```
var matrix=[]; // матриця розміром n*m
for(i=0;i<n;i++){
    matrix[i]=new Array(m); // або matrix[i]=[];
};
```

Масиви є об'єктами і мають численні методи для полегшення роботи з ними.

split (роздільник) дозволяє перетворити рядок в масив, розбивши його по роздільнику.

```
var names = 'Петя, Марина, Василь';
var arr = names.split(','); // arr = ['Петя', 'Марина', 'Василь']
for (var i = 0; i < arr.length; i++) {
    alert (arr [i]);
}
```

Виклик `str.split ("")` розіб'є рядок на літери.

join (роздільник) Протилежний метод до `split`. Він склеює масив в рядок, використовуючи роздільник.

```
var arr = ['Петя', 'Марина', 'Василь'];
alert(arr.join(';')); //Петя;Марина;Василий
```

splice (позиція, лічильник видалення, елемент1, елемент2, ..., елементN) – універсальний метод для роботи з масивами. Вміє все: видаляти елементи, вставляти елементи, замінювати елементи – по черзі й одночасно. Він видаляє вказану кількість елементів починаючи з позиції, а потім вставляє елементи на їх місце. Наступні за видаленими елементами зсуваються, щоб

заповнити їх місце. Допускається використання від'ємного номера позиції, яка в цьому випадку відраховується з кінця.

```
var a=[0,1,2,3,4,5,6,7];
```

```
a.splice (1, 1); // видалити 1 елемент, починаючи з позиції 1,  
a=[0,2,3,4,5,6,7];
```

```
a.splice(-3,2); //видалити 2 елементи, починаючи з 3 від кінця  
позиції, a=[0,2,3,4,7]
```

```
a.splice(0, 3, 5, 5); // видалити 3 елементи на початку і додати  
дві 5, a=[5,5,4,7]
```

```
a.splice(3, 0, 1); // додати без видалення a=[5,5,4,1,7]
```

```
var b=a.splice (1, 2,'видалено 5,4'); //a=[5,'видалено 5,4',1,7];  
b=[5,4];
```

slice (begin, end) копіює ділянку масиву від begin до end, не включаючи end. Вихідний масив при цьому не змінюється.

```
var arr = ["Чому", "треба", "вчити", "JavaScript"];
```

```
var a = arr.slice (1,3); / / елементи 1, 2 (не включаючи 3)  
a=["треба", "вчити"];
```

reverse() змінює порядок елементів у масиві на зворотний.

indexOf (searchElement [, fromIndex]) повертає номер елемента searchElement в масиві, або -1, якщо його немає. Пошук починається з номера fromIndex, якщо він зазначений. Якщо ні – від початку масиву. Для пошуку використовується суворе порівняння ===.

```
var arr = [1, 0, false,1];
```

```
alert (arr.indexOf (0)); // 1
```

```
alert (arr.indexOf (false)); // 2
```

```
alert (arr.indexOf (null)); // -1
```

```
alert (arr.indexOf (1,2)); //3
```

lastIndexOf (searchElement [, fromIndex]) має аналогічні параметри, але шукає останнє входження елемента переглядаючи масив справа-наліво, з кінця або з номера fromIndex, якщо він зазначений.

forEach (функція) викликає функцію для кожного елемента масиву. Функція викликається з параметрами (item, i, arr): item – значення поточного елемента масиву; i – його номер; arr – масив, що перебирається. Зміна значення item не приводить до зміни відповідного елемента, для цього слід звертатись до arr[i].

Приклад. Скласти скрипт, що виводить кожен елемент масиву в окремий рядок у форматі a[i]=...

```
function print(x,i,arr){  
    document.writeln('a['+i+']='+x+'<br>');  
}  
var a=[2,3,5];  
a.forEach(print);  
alert(a);
```

filter (callback) створює новий масив, в який увійдуть тільки ті елементи вихідного масиву, для яких виклик callback (item, i, arr) поверне true.

Приклад. Скласти скрипт, що знаходить всі парні елементи на непарних місцях.

```
function condition(x,i,arr){  
    return (i%2!=0)&&(x%2==0);  
}
```

```
var b=a.filter(condition);
```

map (callback) створює новий масив, який складатиметься із результатів виклику callback (item, i, arr) для кожного елемента arr.

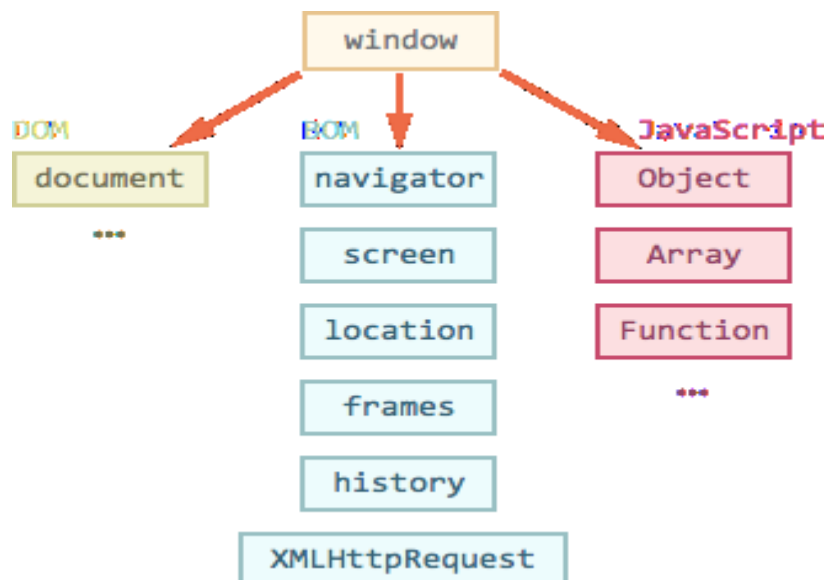
Приклад. Скласти скрипт, що збільшує кожен елемент масиву на його індекс;

```
function change(x,i,arr){  
    return x+i;  
}  
alert(a.map(change));
```

some (callback) повертає true, якщо виклик callback поверне true для якогось елемента масива.

5. БРАУЗЕРНЕ СЕРЕДОВИЩЕ

5.1. Структура браузерних об'єктів



Є “корінний” об’єкт, що називається **window**. Він має дві

ролі:

1. По-перше, це глобальний об'єкт для коду JavaScript у браузері.

2. По-друге, він є “вікном браузера” та надає способи для керування ним.

Приклад:

```
// Відкрити нове вікно/вкладку з URL https://www.uzhnu.edu.ua/  
window.open('https://www.uzhnu.edu.ua/');
```

Document Object Model, або скорочено **DOM**, – представляє весь контент сторінки як об'єкти, які можуть бути змінені.

Об'єкт **document** – це головна “точка входу” до сторінки. Ми можемо змінити або створити що-небудь на сторінці, використовуючи цей об'єкт.

Приклад:

```
// змінити колір фону на червоний  
document.body.style.background = "red";  
// повернути його назад після 1 секунди  
setTimeout(() => document.body.style.background = "", 1000);
```

Модель об'єкта браузера (**BOM**) є додатковими об'єктами, що надані браузером (хост-середовищем) для роботи з усім, крім документа.

Наприклад:

Об'єкт **navigator** забезпечує інформацію про браузер та операційну систему. Є багато його властивостей, але дві найбільш широко відомих: **navigator.userAgent** – інформація про поточний браузер, та **navigator.platform** – про платформу (може допомогти визначити, на якій платформі відкрито браузер –

Windows/Linux/Mac тощо).

Об'єкт **location** дозволяє нам прочитати поточну URL-адресу і може перенаправити веббраузер на нову адресу.

Ось як ми можемо використовувати об'єкт location:

```
alert(location.href); // показує поточний URL-адрес
if (confirm("Перейти на сайт Wikipedia?")) {
    location.href = "https://wikipedia.org"; // перенаправляє
браузер на іншу URL-адресу
}
```

Функції **alert/confirm/prompt** також є частиною **BOM**: вони безпосередньо не пов'язані з документом, але є чистими методами взаємодії з користувачем.

5.2. Дерево DOM

Основним інструментом роботи і динамічних змін на сторінці є **DOM (Document Object Model)** – об'єктна модель, використовувана для XML/HTML-документів.

Згідно з **DOM-моделлю**, документ є ієрархією, деревом. Кожен **HTML**-тег утворює вузол дерева з типом «елемент». Вкладені в нього теги стають дочірніми вузлами. Для представлення тексту створюються вузли з типом «текст».

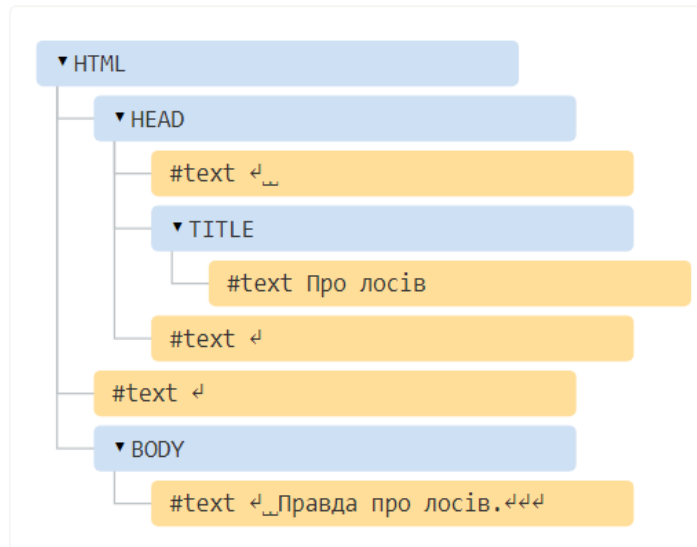
DOM – це подання документа у вигляді дерева об'єктів, доступне для зміни через JavaScript.

Побудуємо, для початку, дерево DOM для наступного документа.

```
<!DOCTYPE HTML>
<html>
<head>
  <h1>Про лосів</h1>
```

```
</head>
<body>
  Правда про лосів.
</body>
</html>
```

Його вигляд:



Теги утворюють **вузли-елементи (element node)**. Текст усередині елементів утворює **текстові вузли (text node)**, позначені як **#text**. Текстовий вузол містить тільки рядок тексту і не може мати нащадків, тобто він завжди на найнижчому рівні. Зверніть увагу на спеціальні символи в текстових вузлах:

- новий рядок: `<br>`
- пробіл: `< >`

Пробіли і нові рядки – це теж текст, повноправні символи, які враховуються в DOM.

Навіть директива **<!DOCTYPE ...>**, яку ми ставимо на початку HTML, теж є DOM-вузлом і знаходиться в дереві DOM безпосередньо перед **<html>**. На ілюстраціях вище цей факт

прихований, оскільки ми з цим вузлом працювати не будемо. Навіть сам об'єкт **document**, формально, є DOM-вузлом, найбільш кореневим.

Усього розрізняють 12 типів вузлів, але на практиці ми працюємо з чотирма з них:

- документ – точка входу в DOM;
- елементи – основні будівельні блоки;
- текстові вузли – містять, власне, текст;
- коментарі – іноді в них можна включити інформацію, яка не буде показана, але доступна з JS.

Можливості, які дає DOM

DOM потрібен для того, щоб маніпулювати сторінкою – читати інформацію з HTML, створювати і змінювати елементи. Вузол HTML можна отримати як *document.documentElement*, а BODY – як *document.body*.

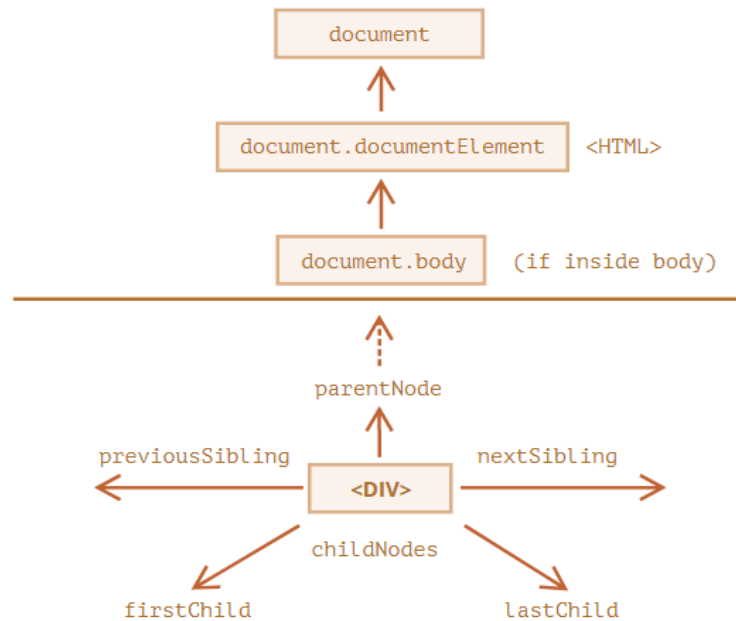
Наприклад, можна поміняти колір BODY і повернути назад:

```
document.body.style.backgroundColor = 'red';  
alert('Поміняли колір BODY');
```

DOM надає можливість робити зі сторінкою все, що завгодно.

5.3. Навігація по DOM-елементах

Основні посилання, за якими можна переходити між вузлами DOM:



`<HTML>` = `document.documentElement`

`<BODY>` = `document.body`

Друга точка входу – `document.body`, яка відповідає тегу `<body>`. У сучасних браузерах також є **`document.head`** – пряме посилання на `<head>`. Є один нюанс: **`document.body` може дорівнювати `null`**. Не можна отримати доступ до елемента, якого ще не існує в момент виконання скрипта. Зокрема якщо скрипт знаходиться в `<head>`, то в ньому недоступний **`document.body`**.

У світі **DOM** як значення, що означає «немає такого елемента» або «вузол не знайдений», використовується не **`undefined`**, а **`null`**.

Дочірні елементи: `childNodes`, `firstChild`, `lastChild`

Дочірні елементи (або діти) – елементи, які лежать безпосередньо всередині даного. Наприклад, усередині `<HTML>`

зазвичай лежать **<HEAD>** і **<BODY>**.

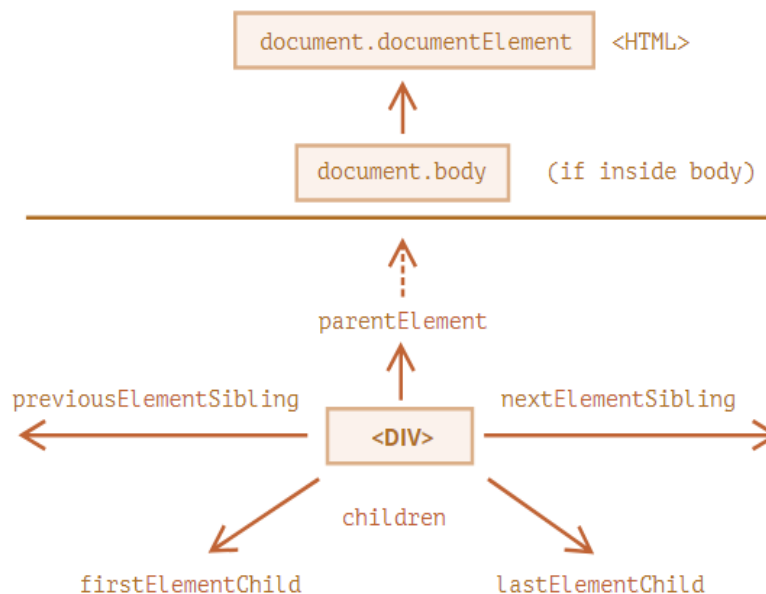
Нащадки – всі елементи, які лежать всередині даного, разом з їхніми дітьми, дітьми їхніх дітей і так далі. Тобто, все піддерево **DOM**.

Псевдо-масив **childNodes** зберігає всі дочірні елементи, включно з текстовими.

Батьківські і сусідні елементи

Доступ до елементів зліва і справа даного можна отримати за посиланнями **previousSibling/nextSibling**. Батько доступний через **parentNode**. Якщо довго йти від одного елемента до іншого, то рано чи пізно дійдеш до кореня DOM, тобто до **document.documentElement**, а потім і **document**.

Навігація тільки за елементами



Ці посилання схожі на ті, що й раніше, тільки в ряді місць стоїть слово **Element**:

- **children** – тільки дочірні вузли-елементи;
- **firstElementChild**, **lastElementChild** – відповідно, перший і останній дочірні елементи;
- **previousElementSibling**, **nextElementSibling** – сусідні елементи;
- **parentElement** – батьківський елемент.

Навіщо **parentElement**? Невже бувають батьки не-елементи?

Властивість **elem.parentNode** повертає батька елемента. Вона завжди однакова з **parentElement**, крім одного винятку:

```
alert(document.documentElement.parentNode); // Document
alert(document.documentElement.parentElement); // Null
```

Завжди вірні рівності:

```
elem.firstElementChild === elem.children [0]
elem.lastElementChild === elem.children [elem.children.length - 1]
```

Пошук: **getElement * і querySelector *** (і не тільки)

Якщо елементу призначений спеціальний атрибут **id**, то можна отримати його викликом **document.getElementById ("ідентифікатор")**.

Приклад:

```
<div id="content"> Виділимо цей елемент </div>
<script>
  var elem = document.getElementById('content');
  elem.style.background = 'red';
  alert( elem == content ); // true
  content.style.background = ""; // Один і той же елемент
</script>
```

Метод **elem.getElementsByTagName(tag)** шукає всі елементи з заданим тегом **tag** всередині елемента **elem** і повертає їх у вигляді

списку. Регістр тега не має значення.

```
// Отримати всі div-елементи  
var elements = document.getElementsByTagName('div');
```

Звернемо увагу: на відміну від **getElementById**, який існує тільки в контексті **document**, метод **getElementsByTagName** може шукати всередині будь-якого елемента.

Наприклад, знайдемо всі елементи **input** всередині таблиці:

Приклад:

```
<table id="age-table">  
  <tr>  
    <td>Ваш вік:</td>  
    <td>  
      <label>  
        <input type="radio" name="age" value="young" checked>  
молодше 18  
      </label>  
      <label>  
        <input type="radio" name="age" value="mature"> від 18  
до 50  
      </label>  
      <label>  
        <input type="radio" name="age" value="senior"> старше  
60  
      </label>  
    </td>  
  </tr>  
</table>  
<script>  
  var tableElem = document.getElementById('age-table');  
  var elements = tableElem.getElementsByTagName('input');  
  for (var i = 0; i < elements.length; i++) {  
    var input = elements[i];  
    alert( input.value + ': ' + input.checked );  
  }  
</script>
```

```
}  
</script>
```

Можна отримати всіх нащадків, передавши зірочку "*" замість тега:

```
// Отримати всі елементи документа  
document.getElementsByTagName('*');  
// Отримати всіх нащадків елемента elem:  
elem.getElementsByTagName('*');
```

Повертається колекція, а не елемент. Колекцію потрібно або перебрати в циклі, або отримати елемент за номером і вже йому привласнювати **value**, наприклад так:

```
document.getElementsByTagName('input')[0].value = 5;
```

getElementsByTagName

Виклик **document.getElementsByTagName(name)** дозволяє отримати всі елементи з даними атрибутом **name**.

Наприклад, всі елементи з ім'ям **age**:

```
var elems = document.getElementsByTagName('age');
```

getElementsByTagName

Виклик **elem.getElementsByTagName(className)** повертає колекцію елементів з класом **className**. Знаходить елемент і в тому випадку, якщо у нього кілька класів, а шуканий – один з них. Підтримується всіма сучасними браузерами.

Приклад:

```
<div class="article">Стаття</div>  
<div class="long article">Довжина статті</div>  
<script>  
  var articles = document.getElementsByTagName('article');  
  alert( articles.length ); // 2, знайде обидва елементи  
</script>
```

Як і **getElementsByTagName**, цей метод може бути викликаний і в контексті **DOM-елемента**, і в контексті документа.

querySelectorAll

Виклик **elem.querySelectorAll (CSS)** повертає всі елементи всередині **elem**, що задовольняють **CSS-селектор CSS**. Він є у всіх сучасних браузерах.

Наступний запит отримує всі елементи **li**, які є останніми нащадками в **ul**:

Приклад:

```
<ul>
  <li>Цей</li>
  <li>тест</li>
</ul>
<ul>
  <li>повністю</li>
  <li>пройдений</li>
</ul>
<script>
  var elements = document.querySelectorAll('ul > li:last-
child');
  for (var i = 0; i < elements.length; i++) {
    alert( elements[i].innerHTML ); // "тест", "пройдений"
  }
</script>
```

querySelector

Виклик **elem.querySelector (CSS)** повертає не всі, а тільки перший елемент, відповідний **CSS-селектору CSS**. Інакше кажучи, результат такий же, як і при **elem.querySelectorAll (CSS) [0]**, але в останньому виклику спочатку шукаються всі елементи, а потім

береться перший, а в **elem.querySelector(CSS)** шукається тільки перший, тобто він ефективніший.

matches

Метод **elem.matches(CSS)** нічого не шукає, а перевіряє, чи задовольняє **elem** селектор **CSS**. Він повертає **true** або **false**. Цей метод буває корисним, коли ми перебираємо елементи (в масиві або за звичайними навігаційним посиланнях) і намагаємося відфільтрувати ті з них, які нам цікаві.

Раніше в специфікації він називався **matchesSelector**, і більшість браузерів підтримують його під цим старим іменем, або з префіксами **ms** / **moz** / **webkit**.

Приклад:

```
<a href="http://example.com/file.zip">...</a>
<a href="http://example.ua">...</a>
<script>
  var elems = document.body.children;
  for (var i = 0; i < elems.length; i++) {
    if (elems[i].matches('a[href$="zip"]')) {
      alert("Посилання на архів: " + elems[i].href);
    }
  }
</script>
```

closest

Метод **elem.closest (CSS)** шукає найближчий елемент вище за ієрархією **DOM**, що підходить під **CSS-селектор CSS**. Сам елемент теж включається в пошук. Інакше кажучи, метод **closest** пробігається від поточного елемента вгору по ланцюжку батьків і перевіряє, чи підходить елемент під вказаний **CSS-селектор**.

Якщо підходить – зупиняється і повертає його. Він найновіший з методів, розглянутих у цьому розділі.

Приклад використання (браузер повинен підтримувати `closest`):

```
<ul>
  <li class="chapter">Розділ I
    <ul>
      <li class="subchapter">Розділ <span
class="num">1.1</span></li>
      <li class="subchapter">Розділ <span
class="num">1.2</span></li>
    </ul>
  </li>
</ul>

<script>
  var numberSpan = document.querySelector('.num');
  // найближчий елемент зверху, який підходить під селектор li
  alert(numberSpan.closest('li').className) // subchapter
  // найближчий елемент зверху, який підходить під селектор
.chapter
  alert(numberSpan.closest('.chapter').tagName) // LI
  // найближчий елемент зверху, який підходить під селектор span
  // це сам numberSpan, так як пошук включає в себе сам
елемент
  alert(numberSpan.closest('span') === numberSpan) // true
</script>
```

nodeName і tagName

Є цілі дві властивості – **nodeName** і **tagName**, – які містять назву (тег) елемента вузла.

Назва HTML-тега завжди знаходиться у верхньому регістрі. Наприклад, для **document.body**:

```
alert(document.body.nodeName); // BODY
```

```
alert(document.body.tagName); // BODY
```

Приклад:

```
<body>
  <!-- коментар -->
  <script>
    // для коментаря
    alert( document.body.firstChild.nodeName ); // #comment
    alert( document.body.firstChild.tagName ); // undefined (в
IE8 - "!")
    // для документу
    alert( document.nodeName ); // #document, так як корінь
DOM - не елемент
    alert( document.tagName ); // undefined
  </script>
</body>
```

При роботі з елементами, як це зазвичай буває, є сенс використовувати властивість **tagName** – вона коротша.

innerHTML: вміст елемента

Властивість **innerHTML** дозволяє отримати **HTML-вміст** елемента у вигляді рядка. В **innerHTML** можна і читати, і писати.

Приклад виведе на екран весь вміст **document.body**, а потім замінить його на інше:

Приклад:

```
<body>
  <p>Параграф</p>
  <div>Div</div>
  <script>
    alert( document.body.innerHTML ); // читаємо поточний
вміст
    document.body.innerHTML = 'Новий BODY!'; // замінюємо
вміст
```

```
</script>
</body>
```

Додавання `innerHTML` + = здійснює перезапис. Якщо в `innerHTML` є тег `script` – він не буде виконаний.

outerHTML: HTML елемента цілком

Властивість `outerHTML` містить `HTML` елемента цілком.

Приклад:

```
<div>Привіт <b>Світ</b></div>
<script>
  var div = document.body.children[0];
  alert( div.outerHTML ); // <div>Привіт <b>світ</b></div>
</script>
```

Змінити `outerHTML` елемента неможливо

nodeValue / data: вміст текстового вузла

Властивість `innerHTML` є тільки у вузлів-елементів.

Вміст текстових вузлів або коментарів на читання і запис через властивість `data`.

Приклад:

```
<body>
  Привіт
  <!-- Коментар -->
  <script>
    for (var i = 0; i < document.body.childNodes.length; i++)
    {
      alert( document.body.childNodes[i].data );
    }
  </script>
  Бувай
</body>
```

Якщо ви запустите цей приклад, то побачите, як виводяться

послідовно:

Привіт – це вміст першого вузла (текстового).

Коментар – це вміст другого вузла (коментаря).

Прогалини – це вміст невеликого пробільного вузла після коментаря до скрипта.

undefined – далі цикл дійшов до `<script>`, але це вузол-елемент, у нього немає `data`.

Загалом після `<script> ... </script>` і до закриття `</body>` в документі є ще один текстовий вузол. Однак на момент роботи скрипта браузер ще не знає про нього, тому не виведе.

Текст: `textContent`

Властивість **`textContent`** містить тільки текст усередині елемента, за винятком усіх `<teгiv>`. **`textContent`** повертає точно весь текст, включно з переведенням рядків і прогалинами, але без тегів. Іншими словами, **`elem.textContent`** повертає конкатенацію всіх текстових вузлів всередині **`elem`**.

У цьому прикладі ім'я відвідувача потрапить у перший **`div`** як **`innerHTML`**, а в другій – як текст:

Приклад:

```
<div></div>
<div></div>
<script>
    var name = prompt("Введіть ім'я", "<b>Вінні-пук</b>");
    document.body.children[0].innerHTML = name;
    document.body.children[1].textContent = name;
</script>
```

Вивід:

Вінні-пух

`<b>Вінні-пух</b>`

При запуску прикладу ми побачимо, що в перший **div** текст від відвідувача вставився в саме як **HTML**, тобто теги стали саме тегами, а в другій – як звичайний текст.

Навряд чи ми дійсно хочемо, щоб відвідувачі вставляли в наш сайт довільний **HTML-код**. Присвоєння через **textContent** – один із способів від цього захиститися.

Властивість **hidden**

Як правило, видимий або невидимий вузол, визначається через **CSS**, властивостями **display** або **visibility**. У стандарті **HTML5** передбачений спеціальний атрибут і властивість для цього: **hidden**. Його підтримують всі сучасні браузері. У прикладі нижче другий і третій `<div>` приховані:

Приклад:

```
<div>Текст</div>
<div hidden>З атрибутом hidden</div>
<div>Із властивістю hidden</div>
<script>
  var lastDiv = document.body.children[2];
  lastDiv.hidden = true;
</script>
```

Технічно атрибут **hidden** працює так само, як **style = "display: none"**. Але його простіше поставити через **JavaScript** (менше літер), і можуть бути переваги для скринрідерів та інших нестандартних браузерів.

Додавання та видалення вузлів

Для створення елементів використовуються такі методи:

```
document.createElement(tag)
```

Створює новий елемент із зазначеним тегом:

```
var div = document.createElement('div');  
document.createTextNode(text)
```

Створює новий текстовий вузол з даним текстом:

```
var textElem = document.createTextNode("Тут був я");
```

Створення повідомлення. У нашому випадку ми хочемо зробити **DOM**-елемент **div**, дати йому класи і заповнити текстом:

```
var div = document.createElement('div');  
div.className = "alert alert-success";  
div.innerHTML = "<strong>Ура!</strong> Ви прочитали це важливе  
повідомлення.";
```

Після цього коду у нас є готовий **DOM**-елемент. Поки що він присвоєний у змінну **div**, але не видно, оскільки ніяк не пов'язаний зі сторінкою.

Додавання елемента: `appendChild`, `insertBefore`

Щоб **DOM**-вузол був показаний на сторінці, його необхідно вставити в **document**. Для цього насамперед потрібно вирішити, куди ми будемо його вставляти. Припустимо, що ми вирішили, що вставляти будемо в якийсь елемент **parentElem**, наприклад `var parentElem = document.body`. Для вставки всередину **parentElem** є такі методи:

parentElem.appendChild (elem)

Додає **elem** у кінець дочірніх елементів **parentElem**.

Наступний приклад додає новий елемент у кінець ****:

Приклад:

```
<ol id="list">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>
<script>
  var newLi = document.createElement('li');
  newLi.innerHTML = 'Привіт, світ!';
  list.appendChild(newLi);
</script>
```

1. 0
2. 1
3. 2
4. Привіт, світ!

parentElem.insertBefore (elem, nextSibling)

Вставляє **elem** у колекцію дітей **parentElem**, перед елементом **nextSibling**.

Наступний код вставляє новий елемент перед другим ****:

Приклад:

```
<ol id="list">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>
<script>
  var newLi = document.createElement('li');
  newLi.innerHTML = 'Привіт, світ!';
  list.insertBefore(newLi, list.children[1]);
</script>
```

Для вставки елемента в початок досить вказати, що вставляти

будемо перед першим нащадком:

```
list.insertBefore(newLi, list.firstChild);
```

У читача, який подивиться на цей код уважно, напевно, виникне питання: «А що якщо **list** взагалі порожній, у цьому випадку, адже **list.firstChild = null**, чи відбудеться вставка?». Відповідь – так, відбудеться. Річ у тім, що якщо другим аргументом вказати **null**, то **insertBefore** спрацює як **appendChild**:

```
parentElem.insertBefore(elem, null);  
// Те ж, що і:  
parentElem.appendChild(elem)
```

Так що **insertBefore** універсальний.

Усі методи вставки повертають вставлений вузол.

Наприклад, **parentElem.appendChild (elem)** повертає **elem**.

Додамо повідомлення в кінець **<body>**:

Приклад:

```
<style>  
.alert {  
  padding: 15px;  
  border: 1px solid #d6e9c6;  
  border-radius: 4px;  
  color: #3c763d;  
  background-color: #dff0d8;  
}  
</style>  
<body>  
  <h3>Моя сторінка</h3>  
</body>  
<script>  
  var div = document.createElement('div');  
  div.className = "alert alert-success";  
  div.innerHTML = "<strong>Ура!</strong> Ви прочитали це
```

```
важливе повідомлення.";
    document.body.appendChild(div);
</script>
```

... А тепер – на початок **<body>**:

Приклад:

```
<style>
.alert {
    padding: 15px;
    border: 1px solid #d6e9c6;
    border-radius: 4px;
    color: #3c763d;
    background-color: #dff0d8;
}
</style>
<body>
    <h3>Моя сторінка</h3>
</body>
<script>
    var div = document.createElement('div');
    div.className = "alert alert-success";
    div.innerHTML = "<strong>Ура!</strong> Ви прочитали це
важливе повідомлення.";
    document.body.insertBefore(div, document.body.firstChild);
</script>
```

Клонування вузлів: **cloneNode**

Виклик **elem.cloneNode (true)** створить «глибоку» копію елемента – разом з атрибутами, включно з піделементами. Якщо ж викликати з аргументом **false**, то копія буде зроблена без дочірніх елементів. Це потрібно набагато рідше.

Приклад зі вставкою копії повідомлення:

Приклад:

```
<style>
```

```

.alert {
  padding: 15px;
  border: 1px solid #d6e9c6;
  border-radius: 4px;
  color: #3c763d;
  background-color: #dff0d8;
}
</style>
<body>
  <h3>Моя сторінка</h3>
</body>
<script>
  var div = document.createElement('div');
  div.className = "alert alert-success";
  div.innerHTML = "<strong>Ура!</strong> Ви прочитали це
важливе повідомлення.";

  document.body.insertBefore(div, document.body.firstChild);
  // створити копію вузла
  var div2 = div.cloneNode(true);
  // копію можуть відредагувати
  div2.querySelector('strong').innerHTML = 'Супер!';
  // вставимо її після поточного повідомлення
  div.parentNode.insertBefore(div2, div.nextSibling);
</script>

```

Зверніть увагу на останній рядок, який вставляє **div2** після **div**:

```
div.parentNode.insertBefore(div2, div.nextSibling);
```

Для вставки нам потрібен майбутній батько. Ми, можливо, не знаємо, де точно знаходиться **div** (або не хочемо залежати від того, де він), але якщо потрібно вставити поруч з **div**, то батьком виразно буде **div.parentNode**. Ми хотіли б вставити після **div**, але методу **insertAfter** немає, є тільки **insertBefore**, тому вставляємо перед його правим сусідом **div.nextSibling**.

Видалення вузлів: `removeChild`

Для видалення вузла є два методи:

```
parentElem.removeChild(elem)
```

Видаляє **elem** зі списку дітей – **parentElem**.

```
parentElem.replaceChild(newElem, elem)
```

Серед дітей **parentElem** видаляє **elem** і вставляє на його місце **newElem**. Обидва ці методи повертають видалений вузол, тобто **elem**. Якщо потрібно, його можна вставити в інше місце **DOM** відразу або в майбутньому. Якщо ви хочете перемістити елемент на нове місце – не потрібно його видаляти зі старого. Всі методи вставки автоматично видаляють елемент зі старого місця і вставляють на нове місце.

Приклад:

```
<div>Перший</div>
<div>Другий</div>
<script>
  var first = document.body.children[0];
  var last = document.body.children[1];
  // немає необхідності в попередньому removeChild(last)
  document.body.insertBefore(last, first); // поміняти місцями
</script>
```

Метод `remove`

У сучасному стандарті є також метод **elem.remove()**, який видаляє елемент безпосередньо, не вимагаючи посилання на батька. Це часто зручніше, ніж **removeChild**. Він підтримується у всіх сучасних браузерах, крім IE11-.

Текстові вузли для вставки тексту

При роботі з повідомленням ми використовували тільки вузли-

елементи і **innerHTML**. Але і текстові вузли теж мають цікаву сферу застосування. Якщо текст для повідомлення потрібно показати саме як текст, а не як HTML, то можна обернути його в текстовий вузол.

Методи для створення вузлів:

document.createElement(tag) – створює елемент

document.createTextNode(value) – створює текстовий вузол

elem.cloneNode(deep) – клонує елемент, якщо **deep == true**, то з усіма нащадками, якщо **false** – без нащадків.

Вставка і видалення вузлів:

parent.appendChild (elem)

parent.insertBefore (elem, nextSibling)

parent.removeChild (elem)

parent.replaceChild (newElem, elem)

Всі ці методи повертають **elem**.

Метод *document.write*

Метод ***document.write (str)*** працює тільки поки HTML-сторінка перебуває в процесі завантаження. Він дописує текст у поточне місце HTML ще до того, як браузер побудує з нього **DOM**.

HTML-документ нижче буде містити 1 2 3.

```
<body>
  1
  <script>
    document.write (2);
  </script>
  3
</body>
```

Також існує метод **document.writeln (str)** – не менш давній, який додає після **str** символ переводу рядка `"\n"`. Методи **document.write** і **document.writeln** пишуть безпосередньо в текст документа, до того як браузер побудує з нього **DOM**, тому вони можуть записати в документ все, що завгодно, будь-які стилі і незакриті теги. Браузер врахує їх при побудові **DOM**, точно так само, як враховує чергову порцію **HTML**-тексту. Технічно, викликати **document.write** можна в будь-який час, однак, коли **HTML** завантажився, і браузер повністю побудував **DOM**, документ стає «закритим». Спроба дописати щось у закритий документ відкриває його заново. При цьому весь поточний зміст видаляється. Наприклад, **document.write** використовують для вставки рекламних скриптів і різних лічильників, коли **URL** скрипта необхідно генерувати динамічно, додаючи в нього параметри з **JavaScript**, наприклад:

```
<script>
    // в url вказано поточну роздільну здатність екрана
    користувача
    var url = 'http://ads.com/buyme?screen=' + screen.width +
    "x" + screen.height;
    // завантажити такий скрипт прямо зараз
    document.write('<script src="' + url + '"></scr' + 'ipt>');
</script>
```

5.4. Основи роботи з подіями

Подія – це сигнал від браузера про те, що щось сталося. Існує багато видів подій.

Події миші

click – виникає, коли клікати на елемент лівою кнопкою миші;
contextmenu – виникає, коли клікати на елемент правою кнопкою миші;
mouseover – виникає, коли на елемент наводиться миша;
mousedown і **mouseup** – коли кнопку миші натиснули або відпустили;
mousemove – при русі миші.

Події на елементах управління

submit – відвідувач відправив форму `<form>`;
focus – відвідувач фокусується на елементі, наприклад натискає на `<input>`.

Клавіатурні події

keydown – коли відвідувач натискає клавішу;
keyup – коли відвідувач відпускає клавішу.

Події документа

DOMContentLoaded – коли **HTML** завантажений і оброблений, **DOM** документа повністю побудований і доступний.

Події CSS

transitionend – коли CSS-анімація завершена.

Призначення обробників подій

Є кілька способів призначити події обробник.

1. Використання атрибуту HTML.

Оброблювач може бути призначений прямо в розмітці, в атрибуті, який називається **on** `<подія>`.

```
<input value = "Натисни мене" onclick = "alert('Клік!')" type =  
"button">
```

Зверніть увагу, для вмісту атрибута **onclick** використовуються одинарні лапки, оскільки сам атрибут знаходиться в подвійних.

Часта помилка новачків в тому, що вони забувають, що код знаходиться всередині атрибута. Запис виду **onclick = "alert("Клік! ")"** з подвійними лапками всередині не буде працювати. Якщо вам дійсно потрібно використовувати саме подвійні лапки, то це можна зробити, замінивши їх на **& quot ;**, тобто так: **onclick = "alert(& quot; Клік! & quot;)"**.

Однак зазвичай цього не потрібно, тому що прямо в розмітці пишуться тільки дуже прості обробники. Якщо потрібно зробити щось складне, то є сенс описати це у функції, і в обробнику викликати вже її саму. Наступний приклад по кліку запускає функцію **countRabbits()**.

Приклад:

```
<!DOCTYPE html>  
<html>  
<head>  
  <meta charset="utf-8">  
  <title>Приклад найпростішого скрипта</title>  
  <script>  
    function countRabbits () {  
      for (var i = 1; i <= 3; i ++) {  
        alert("Кролик номер" + i);  
      }  
    }  
  </script>  
</head>  
<body>
```

```
        <input type = "button" onclick = "countRabbits()" value =  
        "порахувати кроликів!" />  
</body>  
</html>
```

Як ми пам'ятаємо, **атрибут HTML-тега не чутливий до регістру**, тому **ONCLICK** буде працювати так само, як **onClick** або **onCLICK**. Але, як правило, атрибути пишуть в нижньому регістрі: **onclick**.

2. Використання властивості DOM-об'єкта.

Можна призначати обробник, використовуючи властивість **DOM-елемента on <подія>**.

Приклад установки обробника **click**:

Приклад:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Приклад найпростішого скрипта</title>  
    <script>  
      function countRabbits () {  
        for (var i = 1; i <= 3; i ++) {  
          alert("Кролик номер" + i);  
        }  
      }  
    </script>  
  </head>  
  <body>  
    <input id = "elem" type = "button" value = "Натисни мене" />  
    <script>  
      elem.onclick = countRabbits;  
    </script>  
  </body>
```

```
</html>
```

Приклад:

```
<input id = "elem" type = "button" value = "Натисни мене" />
<script>
    elem.onclick = function () {
        alert('Дякую');
    };
</script>
```

Якщо обробник заданий через атрибут, то браузер читає HTML-розмітку, створює нову функцію з вмісту атрибута і записує у властивість **onclick**. Цей спосіб, по суті, аналогічний попередньому. Оброблювач зберігається саме в **DOM**-властивості, а атрибут – лише один із способів його ініціалізації.

Ці два приклади коду працюють однаково:

Приклад:

Тільки HTML:

```
<input type = "button" onclick = "alert('Клік!')" value =
"Кнопка" />
```

HTML + JS:

```
<input type = "button" id = "button" value = "Кнопка" />
<script>
    button.onclick = function () {
        alert('Клік!');
    };
</script>
```

Оскільки **DOM-властивість onclick** у підсумку одна, то призначити більше одного обробника так не можна. У прикладі нижче призначення через **JavaScript** перезапише обробник з атрибуту:

```
<input type = "button" id = "elem" onclick = "alert('До!')"
```

```
value = "Натисни мене" />
<script>
  elem.onclick = function () { // перезапише існуючий обробник
    alert('Після'); // Виведеться тільки це
  };
</script>
```

До речі, обробником можна призначити і вже наявну функцію:

```
function sayThanks () {
  alert('Дякую!');
}
elem.onclick = sayThanks;
```

Якщо обробник непотрібний – його завжди можна прибрати призначенням `elem.onclick = null`.

Доступ до елемента через **this**

У середині обробника події **this** посилається на поточний елемент, тобто на той, на якому він спрацював. Це можна використовувати, щоб отримати властивості або змінити елемент.

У коді нижче **button** виводить свій вміст, використовуючи **this.innerHTML**:

```
<button onclick = "alert(this.innerHTML)">
  Натисни мене
</ button>
```

Часті помилки

Якщо ви тільки починаєте працювати з подіями, зверніть увагу на такі особливості.

Функція повинна бути присвоєна як **sayThanks**, а не **sayThanks()**.

```
button.onclick = sayThanks;
```

Якщо додати дужки, то **sayThanks()** – буде вже результат

виконання функції (а оскільки в ній немає **return**, то в **onclick** потрапить **undefined**). Нам же потрібна саме функція.

А ось у розмітці якраз дужки потрібні:

```
<input type = "button" id = "button" onclick = "sayThanks()" />
```

Цю відмінність просто пояснити. При створенні обробника браузером з атрибуту, він автоматично створює функцію з його вмісту. Тому останній приклад – фактично те ж саме, що:

```
button.onclick = function () {  
    sayThanks (); // Вміст атрибута  
};
```

Використовуйте саме функції, а не рядки.

Призначення обробника рядком **elem.onclick = "alert(1)"** можна іноді побачити в старішому коді. Це буде працювати, але не рекомендовано: можуть бути проблеми при стисненні **JavaScript**. Та загалом передавати код у вигляді рядка щонайменше дивно в мові, яка підтримує **Function Expressions**. Це можливо лише з міркувань сумісності, не робіть так.

Не використовуйте `setAttribute`.

Такий виклик працювати не буде:

```
// При натисканні на body будуть помилки  
// Бо при призначенні в атрибут функція буде перетворена в  
рядок  
  
document.body.setAttribute ('onclick', function () {alert(1)});
```

Регістр DOM-властивості має значення. При призначенні через DOM потрібно використовувати властивість `onclick`, а не `ONCLICK`.

Недолік призначення через властивість.

Фундаментальний недолік описаних вище способів призначення обробника – неможливість повісити кілька обробників на одну подію. Наприклад, одна частина коду хоче при кліку на кнопку робити її підсвіченою, а інша – видавати повідомлення. Потрібно «повісити» в різних місцях два обробники.

При цьому новий обробник буде затирати попередній. Наприклад, наступний код насправді призначає один обробник – останній:

```
input.onclick = function () {alert(1); }  
// ...  
input.onclick = function () {alert(2); } // Замінить попередній  
обробник
```

Розробники стандартів досить давно це зрозуміли і запропонували альтернативний спосіб призначення обробників за допомогою спеціальних методів, які вільні від зазначеного недоліку.

addEventListener і removeEventListener

Методи **addEventListener** і **removeEventListener** є сучасним способом призначити або видалити обробник, і при цьому дозволяють використовувати скільки завгодно будь-яких обробників.

Призначення обробника здійснюється викликом **addEventListener** з трьома аргументами:

```
element.addEventListener (event, handler [, phase]);
```

event

Ім'я події, наприклад, **click**.

handler

Посилання на функцію, яку треба поставити обробником.

phase

Необов'язковий аргумент, «фаза», на якій обробник повинен спрацювати. Цей аргумент рідко потрібен.

Обробники, додані через `on ...`-властивість, нічого не знають про стадії перехоплення, а починають працювати зі спливання.

Щоб зловити подію на стадії перехоплення, потрібно використовувати третій аргумент **`addEventListener`**:

Якщо аргумент `true`, то подію буде перехоплено по дорозі вниз.

Якщо аргумент `false`, то подію буде спіймано при спливанні.

Є події, які не спливають, але які можна перехопити. Бувають події, які можна зловити тільки на стадії перехоплення, а на стадії спливання — не можна.

Видалення обробника здійснюється викликом **`removeEventListener`**:

```
// Передати ті ж аргументи, що були у addEventListener  
element.removeEventListener (event, handler [, phase]);
```

Для видалення потрібно передати саме ту функцію-обробник, яка була призначена.

Ось так `removeEventListener` не спрацює:

```
elem.addEventListener ("click", function () {  
    alert('Спасибі!')  
});  
// ....  
elem.removeEventListener ("click", function () {  
    alert('Спасибі!')  
});
```

У **`removeEventListener`** передана не та ж функція, а інша, з однаковим кодом, але це не важливо.

Ось так правильно:

```
function handler () {  
    alert('Дякую!');  
}  
input.addEventListener ("click", handler);  
// ....  
input.removeEventListener ("click", handler);
```

Якщо функцію не зберегти де-небудь, а просто передати в **addEventListener**, як у попередньому коді, то потім отримати її назад, щоб зняти обробник, буде неможливо. Метод **addEventListener** дозволяє додавати кілька обробників на одну подію одного елемента, наприклад:

Приклад:

```
<input id="elem" type="button" value="Натисни на мене"/>  
  
<script>  
    function handler1() {  
        alert('Дякую!');  
    };  
  
    function handler2() {  
        alert('Дякую ще раз!');  
    }  
    elem.onclick = function() { alert("Привіт"); };  
    elem.addEventListener("click", handler1); // Дякую!  
    elem.addEventListener("click", handler2); // Дякую ще раз!  
</script>
```

Як видно з прикладу вище, можна одночасно призначати обробники і через **DOM-властивість** і через **addEventListener**. Однак щоб уникнути плутанини, рекомендується вибрати один спосіб.

У спеціальних методів є ще одна перевага перед DOM-властивостями. Є деякі події, які не можна призначити через DOM-властивість, але можна через **addEventListener**. Наприклад, такою є подія **transitionend**, тобто закінчення **CSS-анімації**. У більшості браузерів вона вимагає призначення через **addEventListener**. Ви можете перевірити це, запустивши код у прикладі нижче. Як правило, спрацює лише другий обробник, але не перший.

Приклад:

```
<style>
  button {
    transition: width 1s;
    width: 100px;
  }
  .wide {
    width: 300px;
  }
</style>
<button id="elem" onclick="this.classList.toggle('wide');">
  Натисни на мене
</button>
<script>
  elem.ontransitionend = function() {
    alert("ontransitionend"); // не спрацює
  };
  elem.addEventListener("transitionend", function() {
    alert("addEventListener"); // спрацює після закінчення
анімації
  });
</script>
```

6. ЗАВДАННЯ ДЛЯ ЛАБОРАТОРНИХ РОБІТ

6.1. Лабораторна робота №1

Вступ до HTML та CSS – створення вебсайту

1. Створити Web-сайт для презентації партії. Сайт повинен містити меню (список, що дозволяє переходити до різних розділів сайту, містить щонайменше 3 пункти), історію політичної сили, їх логотип, таблицю з результатами соціологічних опитувань, список лідерів політичної сили. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідні особи. Сайт має містити аудіо- та відеоінформацію.

2. Створити Web-сайт для презентації футбольної команди. Сайт повинен містити меню (список, що дозволяє переходити до різних розділів сайту, містить щонайменше 3 пункти), історію команди, її логотип, турнірну таблицю з місцем команди в чемпіонаті, список гравців. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідних гравців. Сайт має містити аудіо- та відеоінформацію.

3. Створити Web-сайт для презентації музичної групи. Сайт повинен містити меню (список, що дозволяє переходити до різних розділів сайту, містить щонайменше 3 пункти), історію групи, їх фотографію, таблицю з графіком гастролей, список учасників. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідних музикантів. Сайт має містити аудіо- та відеоінформацію.

4. Створити Web-сайт для презентації баскетбольної команди. Сайт повинен містити меню (список, що дозволяє переходити до різних розділів сайту, містить щонайменше 3 пункти), історію команди, її логотип, турнірну таблицю з місцем команди в чемпіонаті, список гравців. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідних гравців. Сайт має містити аудіо- та відеоінформацію.

5. Створити Web-сайт для презентації ВНЗ. Сайт повинен містити меню (список, що дозволяє переходити до різних розділів сайту, містить щонайменше 3 пункти), історію НЗ, його логотип, таблицю з кількістю студентів різних форм навчання по факультетах, список винаходів. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідний винахід. Сайт має містити аудіо- та відеоінформацію.

6. Створити Web-сторінку для презентації хокейної команди. Сторінка повинна містити зміст (список, що дозволяє переходити до різних розділів сайту, містить щонайменше 3 пункти), історію команди, її логотип, турнірну таблицю з місцем команди в чемпіонаті, список гравців. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідних гравців. Сайт має містити аудіо- та відеоінформацію.

7. Створити Web-сайт для презентації тенісиста. Сайт повинен містити меню (список, що дозволяє переходити до різних розділів сайту, містить щонайменше 3 пункти), біографію спортсмена, його фотографію, таблицю зі світовим рейтингом по різних версіях,

список виграних трофеїв. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідний трофей. Сайт має містити аудіо- та відеоінформацію.

8. Створити Web-сайт для презентації боксера. Сайт повинен містити меню (список, що дозволяє переходити до різних розділів сайту, містить щонайменше 3 пункти), біографію спортсмена, його фотографію, таблицю зі світовими рейтингами по різних версіях, список виграних трофеїв. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідний трофей. Сайт має містити аудіо- та відеоінформацію.

9. Створити Web-сайт для презентації вченого. Сайт повинен містити зміст (список, що дозволяє переходити до різних розділів сторінки, містить щонайменше 3 пункти, меню), біографію вченого, його фотографію, таблицю з датою публікації основних праць, список винаходів. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідний винахід. Сайт має містити аудіо- та відеоінформацію.

10. Створити Web-сайт для презентації співака. Сайт повинен містити зміст (список, що дозволяє переходити до різних розділів сторінки, містить щонайменше 3 пункти, меню), біографію співака, його фотографію, таблицю з графіком гастролей, список альбомів. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідний альбом. Сайт має містити аудіо- та відеоінформацію.

11. Створити Web-сайт для презентації кандидата в

президенти. Сторінка повинна містити меню (список, що дозволяє переходити до різних розділів сторінки, містить щонайменше 3 пункти), біографію кандидата, його фотографію, таблицю з графіком зустрічей із виборцями, список політичних сил, що підтримують кандидата. Елементи списку повинні бути посиланнями на сторінки, що містять інформацію про відповідні сили. Сайт має містити аудіо- та відеоінформацію.

6.2. Лабораторна робота №2

Створення автобіографії на HTML з підключенням стилів на CSS

1. Створити HTML-сторінку, що буде містити:

а) Автобіографію, що складається щонайменше із трьох розділів (дитинство, школа, ВНЗ). Кожен розділ повинен починатись із заголовка.

б) Зміст, який буде містити переходи на вказані розділи (меню з ефектами наведення).

в) Згадки про батьків зробити гіперпосиланнями на сторінки з відомостями про них.

г) Фотографію.

д) Таблицю успішності за минулий семестр.

2. Створити форму на задану тематику. Форма повинна містити щонайменше 5 груп елементів різного типу, зокрема кнопки, перемикачі, прапорці, випадні списки та ін. Здійснити вирівнювання елементів у групах, та стилізацію за допомогою CSS.

1) Написання електронного листа.

2) Карту призовника військкомату.

3) Купівля квитка.

4) Реєстрація користувача на форумі.

5) Пошук інформації про країну.

6) Додавання інформації про фільм чи серіал.

- 7) Додавання інформації про музичну композицію.
- 8) Пошук книги за заданими параметрами.
- 9) Введення інформації про платіж у банку.
- 10) Пошук спортивних результатів.
- 11) Написання електронного листа.

1. Створити і підключити стильовий файл згідно з такими параметрами:

Варіант 1	1) Автобіографія за лівим краєм, 18 курсивним шрифтом Palatino Linotyp, усі букви у верхньому регістрі, відстань між символами рівна двом розмірам елемента. 2) Зміст і відповідні заголовки зробити різними кольорами. Шрифт Comic Sans MS, 16, не підкреслений, на синьому фоні. 3) В першому розділі встановити вирівнювання за шириною, шрифт Times New Roman 14, одинарний міжрядковий інтервал, зробити так, щоб посилання виділялось як звичайний жирний текст. 4) Третій і другий абзац вирівняти за центром, і розмістити на жовтому фоні, а назву ВНЗ виділити білим підкресленим текстом на синьому фоні. 5) Фоном для всього документу встановити зображення сніжинки, яке повторюється.
-----------	---

Варіант 2	1) Слово Автобіографія за центром, 18 жирним курсивним шрифтом Georgia, усі букви маленькі. 2) Зміст і відповідні заголовки зробити синіми. Шрифт Comic Sans MS, 16, виділені лінією зверху. 3) У першому і третьому розділі встановити вирівнювання за правим краєм, шрифт Times New Roman 14, подвійний міжрядковий інтервал, зробити так, щоб посилання виділялось як звичайний курсивний текст. Відстань між словами рівна двом розмірам елемента. 4) У другому абзаці шрифт такий же, вирівняти за центром, виступ абзацу на 1 см, а рік вступу і закінчення школи виділити червоним текстом більшого розміру. 5) Фоном для всього документу встановити емблему ВНЗ, розміщену за центром сторінки, має прокручуватися разом із вмістом сторінки.
Варіант 3	1) Слово Автобіографія за центром, 25 потоншеним шрифтом Comic Sans MS, капітелом, відстань між символами зменшити на 25%. 2) Зміст і відповідні заголовки зробити похилим білим тестом на різнокольорових фонах. 3) У першому і другому розділі встановити вирівнювання за правим краєм, шрифт Verdana 10, подвійний міжрядковий інтервал, зробити так, щоб посилання

	виділялось тільки верхнім регістром всіх букв. 4) Третій абзац вирівняти за шириною, а назву ВНЗ виділити сірим текстом, що блимає. 5) Додати фонове зображення емблеми ВНЗ, що повторюється по вертикалі і розміщена на правому боці сторінки.
Варіант 4	1) Слово Автобіографія за центром, 20 курсивним шрифтом Courier New, всі букви у верхньому регістрі, відстань між буквами рівна половині розміру елемента. 2) Зміст і відповідні заголовки зробити синіми. Шрифт Comic Sans MS, 16, виділені лініями зверху і знизу. 3) В першому розділі встановити вирівнювання за шириною, шрифт Helvetica 12, подвійний міжрядковий інтервал, зробити так, щоб посилання виділялось тільки жирним курсивом. 4) Третій і другий абзац вирівняти за центром і розмістити на жовтому фоні. 5) Додати фонове зображення емблеми ВНЗ, що повторюється вертикально і розміщена на лівому боці сторінки
Варіант 5	1) Автобіографія за центром, 22 жирним шрифтом Palatino Linotyp, всі букви у нижньому регістрі. 2) Зміст і відповідні заголовки зробити різними

	кольорами. Шрифт Comic Sans MS, 16, не підкреслений, на градієнтному фоні. 3) В першому розділі встановити вирівнювання за шириною, шрифт Times New Roman 14, подвійний міжрядковий інтервал, зробити так, щоб після відвідування посилання не виділялось із тексту. 4) Третій і другий абзац вирівняти за центром, і розмістити на фоні емблеми ВНЗ, а назву школи виділити нахилом. 5) Документ розмістити на блакитному фоні.
Варіант 6	1) Слово Автобіографія за центром, 25 похилим шрифтом Comic Sans MS, капітелом 2) Зміст і відповідні заголовки зробити різними кольорами. Шрифт Arial, 16, Закреслений. 3) В першому і третьому розділі встановити вирівнювання за правим краєм, шрифт Times New Roman 14, подвійний міжрядковий інтервал, зробити так, щоб посилання виділялось як звичайний курсивний текст. Відстань між словами збільшити в 1,5 рази. 4) Другий абзац вирівняти за центром, і розмістити на жовтому фоні, а рік закінчення школи виділити білим підкресленим текстом на чорному фоні. 5) Фоном для всього документу встановити емблему ВНЗ, розміщену у верхньому правому куті, має

	прокручуватися разом із вмістом сторінки.
Варіант 7	1) Слово Автобіографія за центром, 18 жирним курсивним шрифтом Georgia, всі букви маленькі, відстань між символами 20 пікселів. 2) Зміст і відповідні заголовки зробити синіми. Шрифт Comic Sans MS, 16, виділені лінією зверху. 3) В першому і третьому розділі встановити вирівнювання за правим краєм, шрифт Times New Roman 14, подвійний міжрядковий інтервал, зробити так, щоб посилання виділялось як звичайний курсивний текст. 4) В другому абзаці шрифт такий же, вирівняти за центром, виступ абзацу на 1 см, а рік вступу і закінчення школи виділити червоним текстом більшого розміру. 5) Документ розмістити на помаранчевому фоні.
Варіант 8	1) Автобіографія за центром, 22 жирним шрифтом Book Antiqua, всі букви у нижньому регістрі. 2) Зміст і відповідні заголовки зробити похилим білим тестом на різнокольорових фонах. 3) В першому і другому розділі встановити вирівнювання за правим краєм, шрифт Verdana 10, подвійний міжрядковий інтервал, зробити так, щоб посилання після відвідування виділялось верхнім регістром всіх букв.

	4) Третій абзац вирівняти за шириною, а назву ВНЗ виділити текстом, що блимає. 5) У центрі кожного абзацу додати зображення прапора України.
Варіант 9	1) Автобіографія за лівим краєм, 18 курсивним шрифтом Palatino Linotyp, всі букви у верхньому регістрі, блимають. 2) Зміст і відповідні заголовки зробити курсивним білим тестом на різнокольорових фонах. 3) В першому розділі встановити вирівнювання за шириною, шрифт Helvetica 12, відступ абзацу на половину сторінки, зробити так, щоб посилання виділялось меншим розміром шрифту. 4) Третій і другий абзац вирівняти за центром, і розмістити на жовтому фоні. 5) Додати фонове зображення емблеми ВНЗ, що повторюється по горизонталі, за центром сторінки і прокручується разом з її вмістом.
Варіант 10	1) Автобіографія за центром, 22 жирним шрифтом Palatino Linotyp, всі букви у верхньому регістрі. 2) Зміст і відповідні заголовки зробити синіми. Шрифт Comic Sans MS, 16, виділені лініями зверху і знизу. 3) В першому розділі встановити вирівнювання за шириною, шрифт Times New Roman 14, кожне слово

	починалось з великої букви. Зробити так, щоб після відвідування посилання не виділялось із тексту. 4) Третій і другий абзац вирівняти за центром, і розмістити на фоні повторюючихся сніжинок а назву школи виділити нахилом. 5) Документ розмістити на фоні прапора України.
Варіант 11	1) Автобіографія за лівим краєм, 18 курсивним шрифтом Palatino Linotyp, всі букви у верхньому регістрі, блимають. 2) Зміст і відповідні заголовки зробити синіми. Шрифт Comic Sans MS, 16, виділені лініями зверху і знизу. 3) В першому розділі встановити вирівнювання за шириною, шрифт Helvetica 12, відступ абзацу на половину сторінки, зробити так, щоб посилання виділялось меншим розміром шрифту. 4) Третій абзац вирівняти за шириною, а назву ВНЗ виділити текстом, що блимає. 5) Додати фонове зображення емблеми ВНЗ, що повторюється по горизонталі, за центром сторінки і прокручується разом з її вмістом

6.3. Лабораторна робота №3

Робота з системою контролю версій GIT

Створити проект(може бути лабораторна робота №1). Ініціалізувати git, створити 3 вітки: master, dev, feature, які мають відрізнятися хоча б на один коміт. Завантажити даний проект до репозиторію на <https://github.com/>. Вміти виконувати всі дії по роботі з GIT описані у розділі 3. GIT.

6.4. Лабораторна робота №4

Робота з JavaScript. Умовні оператори та конструкції, цикли

1. Обчислити значення функції при значеннях змінних **A**, **B**, **C**, **N**, **X**, **Y**, які вводяться користувачем (через **prompt()** (можна і через інпут)). Результат вивести або на сторінку, або через **alert()**.

Варіант 1	$y = \begin{cases} \cos(A + C + N), & \text{якщо } A = C = N, \\ \cos(A \cdot C \cdot N), & \text{якщо } A < C = N, \\ \cos((A + C) \cdot N), & \text{якщо } A < C < N, \\ 0, & \text{в інших випадках.} \end{cases}$
Варіант 2	$y = \begin{cases} \ln x - n, & \text{якщо } x < n, \\ \ln x - n, & \text{якщо } x = n, \\ \cos(nx), & \text{якщо } x > n. \end{cases}$
Варіант 3	$y = \begin{cases} \cos(x^2 + \ln(x)), & \text{якщо } x^2 + \ln(x) > 0 \\ \frac{1}{(x^2 + \ln(x))}, & \text{якщо } x^2 + \ln(x), \\ \cos(x), & \text{якщо } x^2 + \ln(x). \end{cases}$
Варіант 4	$y = \begin{cases} 1, & \text{якщо } a < b < c, \\ 2, & \text{якщо } a = b = c, \\ 3, & \text{якщо } b < a < c, \\ 4, & \text{якщо } b < a < c, \\ 0, & \text{в інших випадках.} \end{cases}$
Варіант 5	$z = \begin{cases} ye^x, & \text{якщо } y < x, \\ y \cdot x, & \text{якщо } y = x, \\ x \cdot e^y, & \text{якщо } y > x. \end{cases}$

Варіант 6	$y = \begin{cases} 1, & \text{якщо } 0 \leq x < 5, \\ 2, & \text{якщо } 5 \leq x < 8, \\ 3, & \text{якщо } x < 0, \\ 4, & \text{якщо } x \geq 8. \end{cases}$
Варіант 7	$y = \begin{cases} 1, & \text{якщо } 2x^2 - x - 3 = 0, \\ 2, & \text{якщо } 2x^2 - x - 3 > 0, \\ 0, & \text{якщо } 2x^2 - x - 3 < 0. \end{cases}$
Варіант 8	$y = \begin{cases} 0, & \text{якщо } 0 \leq n < 5, \\ 1, & \text{якщо } 5 \leq n < 10, \\ 2, & \text{якщо } 10 \leq n < 15, \\ 3, & \text{в інших випадках.} \end{cases}$
Варіант 9	$y = \begin{cases} e^t \cdot \sin(t), & \text{якщо } t < 0 \\ e^t \cdot \cos(t), & \text{якщо } t \geq 0 \end{cases}$ $t = \begin{cases} \cos(x), & \text{якщо } x < 2, \\ \ln(x), & \text{якщо } x \geq 2. \end{cases}$
Варіант 10	$z = \begin{cases} \ln(x) - \operatorname{tg}(x), & 1 \leq x \leq 3, \\ \operatorname{tg}(x), & 3 < x \leq 4, \\ 0, & \end{cases}$
Варіант 11	$y = \begin{cases} (\cos(x))^n, & \text{якщо } n = 1, \\ x^n, & \text{якщо } n = 2, \\ n^x, & \text{якщо } n = 3, \\ 0, & \text{в інших випадках.} \end{cases}$

2. Користувач вводить числа, поки не натисне відміну (кількість чисел, яку введе користувач, невідома). Знайти:

- 1) кількість парних;
- 2) добуток непарних;
- 3) суму двоцифрових;

- 4) середнє арифметичне додатних;
- 5) найбільше від'ємне;
- 6) кількість більших за перше число;
- 7) середнє геометричне додатних;
- 8) найменше з них;
- 9) кількість трицифрових;
- 10) добуток кратних 5;
- 11) найбільше з них.

Результат вивести або на сторінку, або через alert().

3. Користувач вводить число. Потрібно вивести початкове та отримане значення.

- 1) двоцифрове натуральне число > 80 збільшити на 7;
- 2) двоцифрове натуральне число < 30 збільшити у 5 разів;
- 3) натуральне число < 20 збільшити на 10;
- 4) двоцифрове натуральне парне число збільшити у 5 разів;
- 5) двоцифрове натуральне число > 30 збільшити на 10;
- 6) натуральне число < 20 збільшити у 9 разів;
- 7) двоцифрове натуральне непарне число піднести до квадрата;
- 8) трицифрове натуральне число кратне 7 зменшити в 7 разів;
- 9) двоцифрове натуральне число > 15 збільшити у 9 разів;
- 10) трицифрове натуральне число зменшити на 100;
- 11) двоцифрове натуральне число < 20 збільшити у 5 разів.

Результат вивести або на сторінку, або через alert().

4. Обчислити. Значення А, N вводяться користувачем.

Варіант 1	Дано $a \in R, n \in N$. Знайти $\frac{2}{a} + \frac{4}{a(a+1)} + \dots + \frac{2n}{a(a+1)\dots(a+n)}.$
Варіант 2	Дано $a \in R, n \in N$. Знайти $\frac{1}{a} + \frac{1}{a(a+1)} + \dots + \frac{1}{a(a+1)\dots(a+n)}.$
Варіант 3	Дано $a \in R, n \in N$. Знайти $\underbrace{\frac{1!}{a^2} + \frac{2!}{a^3(a+1)} + \frac{3!}{a^4(a+1)^2(a+2)} + \frac{4!}{a^5(a+1)^3(a+2)^2(a+3)} + \dots}_{n \text{ доданків}}$
Варіант 4	Дано дійсне число a і натуральне число n . Обчислити $a(a+1)\dots(a+n-1)$.
Варіант 5	Дано дійсне число a і натуральне число n . Обчислити $\ln a^n + \ln a^{n-1} + \dots + \ln a $.
Варіант 6	Дано дійсне число a і натуральне число n . Обчислити $a^2(a+1)^2 \dots (a+n-1)^2$.

Варіант 7	Дано $x, a \in R, n \in N$. Знайти $\underbrace{(((x+a)^2 + a)^2 + \dots + a)^2 + a}_n.$
Варіант 8	Дано натуральне число n . Обчислити $s = \prod_{i=1}^n i^{i-1}$.

Варіант 9	Дано $a_0 = 4$. Обчислити a_{12} , якщо $a_i = a_{i-1} - \cos\left(\frac{a^i}{12}\right).$
Варіант 10	Дано натуральне число n . Обчислити $s = (2^2 + 4^2 + \dots + (2n)^2) + (3^3 + 5^3 + \dots + (2n+1)^3).$
Варіант 11	Дано натуральне число n і дійсне число x . Обчислити $\cos x + \cos^2 x + \dots + \cos^n x$

Результат вивести або на сторінку, або через alert().

6.5. Лабораторна робота №5

Робота з JavaScript. Масиви та їх використання

1. Дані вводу і виводу відобразити на сторінці через document.write() у вигляді таблиці. Дані вводяться користувачем.

Варіант 1	Дано n дійсних чисел: $x_1, x_2, \dots, x_n$. Знайти середнє арифметичне значення цих чисел.
Варіант 2	Дано n дійсних чисел: $x_1, x_2, \dots, x_n$. Знайти середнє геометричне значення цих чисел.
Варіант 3	Дано n дійсних чисел: $x_1, x_2, \dots, x_n$. Знайти найменше по модулю.

Варіант 4	Дано n дійсних чисел: $x_1, x_2, \dots, x_n$. Знайти число, що має найбільшу дробову частину.
Варіант 5	Дано n дійсних чисел: $x_1, x_2, \dots, x_n$. Знайти найбільше серед від'ємних.
Варіант 6	Дано n дійсних чисел: $x_1, x_2, \dots, x_n$. Знайти найменше серед додатних.
Варіант 7	Дано одновимірний масив дійсних чисел X . У цьому масиві поміняти місцями елементи, що розташовані симетрично відносно середини.
Варіант 8	Знайти добуток від'ємних елементів вектора $a \in R^n$.
Варіант 9	Знайти кількість від'ємних елементів у векторі $x \in R^n$, які розташовані після першого додатного.
Варіант 10	Знайти найбільший елемент серед елементів вектора $x \in R^n$ з парними індексами.
Варіант 11	У векторі $x \in R^n$ всі елементи, які більші за середнє арифметичне замінити нулями.

2. Описати прямокутну матрицю. Розмірність матриці вводиться користувачем. Елементи розрахувати за формулою. Отриманий результат вивести у вигляді матриці. Передбачити можливість вводу матриці користувачем. (Дані вводяться в інпуті).

$$1) \quad e^i - j.$$

$$2) \quad \cos\left(\frac{\pi}{i + j + 1}\right).$$

$$3) \quad \ln(i+1) - j/n.$$

$$4) \quad \sin(\pi j/n).$$

$$5) \quad \operatorname{tg}(i+1) - j.$$

$$6) \quad \sqrt{2} \sin(ij).$$

$$7) \quad n - (i+j).$$

$$8) \quad (i-j)^{n/5}.$$

$$9) \quad \sin i + \cos j.$$

$$10) \quad j * e^i.$$

$$11) \quad (j-j)/(i+j+1).$$

3. Вивести матрицю, у яку можна вводити дані через теги input (розмірність матриці вводиться користувачем). На основі введених даних обчислити і вивести результат:

- 1) Визначити добуток від'ємних елементів матриці з обома непарними індексами.
- 2) Визначити добуток додатних елементів матриці вище головної діагоналі.
- 3) Визначити суму від'ємних елементів матриці з парною сумою індексів.
- 4) Визначити суму додатних елементів матриці з непарною сумою індексів.
- 5) Визначити добуток додатних парних елементів матриці.
- 6) Визначити добуток додатних елементів матриці нижче головної діагоналі.
- 7) Визначити кількість від'ємних елементів матриці з обома парними індексами.
- 8) Визначити кількість додатних елементів матриці з першим парним і другим.
- 9) Визначити кількість від'ємних елементів матриці з обома непарними індексами.
- 10) Визначити кількість додатних елементів матриці вище головної діагоналі.
- 11) Визначити суму від'ємних елементів матриці на побічній діагоналі.

ПЕРЕЛІК ПИТАНЬ ДЛЯ САМОКОНТРОЛЮ

1. Що таке вебсторінка та вебсайт?
2. Які елементи використовують для форматування тексту?
3. Які відмінності між блочними та рядковими елементами?
4. Що таке текстові гіперпосилання та для чого їх використовують?
5. Як задаються таблиці на вебсторінці?
6. Як задаються іфрейми на вебсторінці?
7. Як створити карту посилань?
8. Як задати мультимедіа на вебсторінці?
9. Для чого використовують форми та які атрибути є у форм?
10. Як і для чого використовують такі теги: `<input>`, `<textarea>`, `<select>`?
11. Що таке каскадні таблиці стилів?
12. Які способи зв'язку каскадних стилів із **HTML**-документом існують?
13. Що таке спадкування?
14. Що таке каскадування?
15. Які існують селектори?
16. Що таке клас та яке його застосування?
17. Що таке ідентифікатор та яке його застосування?
18. Що таке контекстний селектор?
19. Яку пріоритетність мають селектори?
20. Що таке псевдокласи та які є їхні різновиди?
21. Які є **CSS**-властивості для стилізації тексту?
22. Що таке позиціонування елементів у **CSS**?
23. Що означають такі **CSS**-властивості: **position**, **float**, **border**, **margin**, **padding**?
24. Які є способи додавання скриптів на сторінку?
25. Які є типи даних у **JS**?
26. Що означає директива **use strict**?
27. Як перевірити, що змінна рівна **NaN**?
28. Порівняйте ключові слова **var**, **let**, **const**.

29. Як працюють оператори присвоєння / порівняння / рядкові / арифметичні / побітові / логічні тощо?
30. Умовні оператори у **JS** та їх використання.
31. Циклічні конструкції у **JS**.
32. Назвіть методи масивів та покажіть, для чого вони потрібні.
33. Рядки у **JS** та методи рядків.
34. Яка різниця між декларацією функції (**function declaration**) та функціональним виразом (**function expression**)?
35. Що таке **DOM**?
36. Як здійснюється навігація по **DOM**-елементах?
37. Як здійснюється пошук елементів на сторінці?
38. Що означають такі властивості як: **innerHTML**, **outerHTML**, **data**, **textContent**?
39. Як здійснюється додавання та видалення вузлів у **JS**?
40. Різновиди події у **JS**.
41. Які є способи призначити обробник події в **JS**?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стівен Шафер. HTML, XHTML та CSS. Біблія користувача, 5-е видання = HTML, XHTML, та CSS Bible, 5th Edition. - М.: "Діалектика", 2011. - 656 с. - ISBN 978-5-8459-1676-1.
2. Вступ до вебтехнологій : навч. посіб. [для студентів спец. 122 – «Комп'ютерні науки»] / розробники : В. І. Пецко, А. С. Беца, Г. В. Мазютинець, О. В. Міца ; рец. : Ф. Е. Гече , Й. І. Головач ; М-во освіти і науки України, ДВНЗ «Ужгородський національний університет», Ф-т інформ. технологій, Каф. інформ. управляючих систем та технологій. — Ужгород : РІК-У, 2023. — 252 с. : іл.
3. Посібник з HTML5 і CSS3 [Електронний ресурс]. - 2021. - Режим доступу до ресурсу: <https://metanit.com/web/html5/>.
4. Kotulenko V. S. Development of a Web application for teaching the basics of HTML and CSS: dis. – Siberian Federal University; Khakass Technical Institute—branch of Siberian Federal University, 2021.
5. Сімакова О. Каскадні таблиці стилів. Стильове оформлення сторінок [Електронний ресурс] / Олена Сімакова. – 2023. – Режим доступу до ресурсу: <https://sites.google.com/view/computer-science-teach/>.
6. Базовий синтаксис CSS [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: <https://html-css.co.ua/self-css/bazoviy-sintaksis-css/>.
7. GIT Tutorial [Електронний ресурс]. - 2023. - Режим доступу до ресурсу: <https://www.w3schools.com/git/>.
8. GIT Documentation [Електронний ресурс]. - 2023. - Режим доступу до ресурсу: <https://git-scm.com/docs>.
9. Сучасний підручник JavaScript [Електронний ресурс]. - 2023. - Режим доступу до ресурсу: <https://uk.javascript.info/>.
10. JavaScript Tutorial [Електронний ресурс]. - 2023. - Режим доступу до ресурсу: <https://www.w3schools.com/js/>.

Електронне навчальне видання

В.І. Пецко

**Основи веброзробки
та вебпрограмування**

Електронний навчальний посібник
для студентів спеціальності F3 «Комп'ютерні науки»

Формат 60x84/8.
Гарнітура Times New Roman
Ум.друк.арк. 20,34. Зам. № 41.

Редакційно-видавничий відділ ДВНЗ «УжНУ».
88000, м. Ужгород, вул. Заньковецької, 89.
E-mail: dep-editors@uzhnu.edu.ua

Видавництво УжНУ «Говерла».
88000, м. Ужгород, вул. Капітульна, 18.
*Свідоцтво про внесення до державного реєстру
видавців, виготівників і розповсюджувачів видавничої продукції
Серія 3т № 32 від 31 травня 2006 року*