
**В. А. Каплун, М. С. Ціхоцький, В. В. Лукічов**

# ОСНОВИ WEB-ПРОГРАМУВАННЯ

## Теорія і практика

Міністерство освіти і науки України
Вінницький національний технічний університет

**В. А. Каплун, М. С. Ціхоцький, В. В. Лукічов**

# ОСНОВИ WEB-ПРОГРАМУВАННЯ

## Теорія і практика

**Електронний збірник завдань
комбінованого (локального та мережного) використання**

Вінниця
ВНТУ
2023

УДК 004.432
К20

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 15 від 29.06.2023 р.)

Рецензенти:

**Романюк О. Н.,** доктор технічних наук, професор
**Штовба С. Д.,** доктор технічних наук, професор
**Майданюк В. П.,** кандидат технічних наук, доцент

**Каплун, В. А**

**К20** Основи web-програмування. Теорія і практика : електронний навчальний посібник комбінованого (локального та мережного) використання [Електронний ресурс] / Каплун В. А., Ціхоцький М. С., Лукічов В. В. – Вінниця : ВНТУ, 2023. – 128 с.

Навчальний посібник призначений для надання майбутнім фахівцям теоретичних знань і практичних навичок, необхідних для вирішення питань, пов’язаних з проектуванням та розробкою веб-сайтів у глобальній мережі інтернет з використанням сучасних інструментальних засобів: HTML, CSS, JavaScript.

Посібник містить теоретичний матеріал, перелік додаткових інформаційних джерел з кожної теми, велику кількість прикладів і рекомендацій, а також перелік індивідуальних завдань до виконання практичних і лабораторних робіт.

УДК 004.432

**ЗМІСТ**

# ВСТУП

Стрімкий розвиток мережі Internet за останні роки привів до збільшення кількості нових онлайн-сервісів. Починаючи від первісних характеристик і основних сервісів, доступних користувачам (пошта і веб), Internet розвинувся у велику всесвітню павутину з великою кількістю різних сервісів, технологічних рішень і став тим місцем, де зустрічаються практично бізнес-організації всього світу.

Спостерігається зростання кількості компаній, які переходять на електронний бізнес. Вони відкривають свої корпоративні домени, створюють свої веб-сайти, рекламують свої товари і послуги в інтернеті, запускають продажі онлайн.

Основна роль в технічній частині цієї роботи належать саме веб-програмістам і розробникам. Їх завдання:

- створювати веб-сайти для різних цілей і виконувати html верстку;
- з'єднувати їх з відповідною базою даних;
- писати веб-сервлети і створювати веб-сервіси;
- створювати технічні передумови для продажів онлайн і просування товарів і послуг;
- дбати про безпеку веб-сайтів і веб-серверів;
- оптимізувати існуючі веб-додатки клієнтів;
- робити можливою автоматизацію публікації контенту.

Веб-програмування – це створення сайтів і програм, які працюють у мережі.

Внаслідок засвоєння теоретичного матеріалу і виконання практичних робіт студенти вмітимуть створювати сайти з використанням HTML, CSS та JavaScript: застосовувати основні правила побудови документів HTML; застосовувати основні властивості каскадних таблиць стилів; використовувати засоби каскадних таблиць стилів для оформлення сторінки; використовувати основні об’єкти браузера та власно визначені об'єкти JavaScript; розробляти нескладні сценарії обробки подій; створювати серверні сценарії та забезпечувати їх взаємодію з базами даних та веб-сервісами.

Після засвоєння теоретичного матеріалу з кожної теми пропонується виконати практичну роботу і підготувати по ній звіт (додаток А).

Основні технології і засоби, які будуть використовуватись під час створення web-сайтів, такі:

- HTML – відповідає за структурування інформації та виділення логічних блоків на web-сторінках сайту;
- CSS – використовується для оформлення та позиціонування, подання інформації на web-сторінках;
- JavaScript – використовується для реалізації динаміки і інтерактивної взаємодії з користувачем.

# 1 ОСНОВИ HTML. РОЗРОБЛЕННЯ СТРУКТУРИ WEB-СТОРІНКИ МОВОЮ HTML

## Історія та етапи становлення мови HTML

Більшість сучасних інтернет технологій базується на давно використовуваній мові HTML, що найбільше дискутується. Вона була розроблена для виконання розмітки та оформлення документів, що розміщуються на веб-сторінках. Свої перші риси мова почала набувати 1986 року. Поштовхом стало ухвалення Міжнародною організацією зі стандартизації (ISO) ISO-8879-стандарту – Standard Generalized Markup Language (SGML). До нього додавався опис, у якому йшлося про те, що SGML призначений **для структурної розмітки тексту**. Опис зовнішнього вигляду документа не передбачався. Виходячи з цього, можна зробити висновок про те, що SGML не був системою для розмітки тексту і не мав будь-якого списку структурних елементів мови, які використовуються в певних умовах. Мова передбачала опис синтаксису написання основних елементів розмітки. Через деякий час вони отримали добре відому сьогодні назву – «теги».

Незважаючи на те, що мова SGML, як і інші схожі програми, не отримала особливого розвитку, але й не була остаточно забута. У 1991 році Європейський інститут фізики частинок оголосив необхідність розробки механізму, що дозволить передавати гіпертекстову інформацію через Глобальну мережу. Саме SGML ліг в основу майбутньої мови HTML.

У своєму розвитку HTML пройшла такі етапи становлення.

*І етап*. Близько сорока тегів містила HTML 1.2. Опису фізичного подання документів все одно не було. Як і її прабатько – SGML, вона була переважно орієнтована на логічну та структурну розмітку тексту. Втім, деякий натяк на те, як буде фізично подана сторінка, ряд тегів все ж таки робив.

*ІІ етап*. Розробкою HTML 2.0 зайнявся консорціум W3C. Перший результат вдалося отримати через рік насиченої роботи – у 1995 році. Практично паралельно обговорювалися можливості версії 3.0. Якщо другу версію не можна назвати відмінною від першої, то третя стала безумовним проривом.

HTML 3.0 містила цікаві новинки:

- розмітку математичних формул;
- теги для створення сторінок;
- вставлення рисунків, що обтікаються текстом та інше.

Однак, цього було недостатньо, потреба у візуальному оформленні гіпертекстових сторінок ставала все більш актуальною. Тоді W3C приступили до створення самостійної системи, яка не суперечить основам HTML, але дозволяє описувати візуальне оформлення документів. Результатом стала поява CSS – Cascading Style Sheets – ієрархічні стильові специфікації, наділені унікальним синтаксисом, структурою, завданнями.

Істотне розширення тегів відбулося з подачі Netscape Communications – корпорації, яка запустила перший комерційний браузер – Netscape Navigator. Нововведення мали лише одну мету – покращити зовнішній вигляд документа, але водночас вони абсолютно суперечили споконвічним принципам мови.

*III етап*. HTML версії 3.2 створили у найкоротші терміни. Він був орієнтований на Microsoft Internet Explorer. Донедавна ця версія HTML була єдиним стандартом мови розробки інтернет-проектів. Однак, напрямок розвивається дуже активно, за допомогою HTML вдалося надати якусь упорядкованість елементам розмітки всіх браузерів, але можливостей мови ставало недостатньо.

*IV етап*. У 2004 році прийняли нову версію HTML – 4.01. Вона забезпечила відмінні показники крос-браузерності та крос-платформеності. Але з 1998 року консорціум W3C все-таки не полишає спроб замінити його мовою на основі мови XML – XHTML 1.0. Далі йде непростий шлях вдосконалення XHTML1 до XHTML2.

*V етап*. Приблизно в той самий час (починаючи з 2004 р.) група розробників почала розглядати майбутнє Всесвітньої павутини в іншому ракурсі. Замість спроб розібратися, що було неправильним в HTML, вони сфокусувалися на тому, чого в ньому не вистачало, що хотіли б мати веб-розробники для втілення своїх ідей.

Передбачається, що число 5 у назві HTML5 означає: цей стандарт є продовженням стандарту HTML. Це, звичайно, не зовсім правильно, тому що HTML5 підтримує всі розробки, що існували в області створення веб-сторінок протягом десяти років після випуску HTML 4.01, включно й строгий синтаксис в стилі XHTML, а також багато інновацій для JavaScript. Проте ця назва робить ясним таке: мова HTML5 може підтримувати угоди XHTML, але потребує дотримання правил HTML.

Ознайомитись з офіційною версією стандарту HTML5 організації W3C можна на веб-сайті за адресою: www.w3.org/TR/html5.

Сьогодні все частіше використовується CSS, але й можливості HTML, незважаючи на свої недоліки, істотно розширилися. Вона залишається мовою логічної розмітки гіпертексту, тобто, не пов'язаною з оформленням документа. Сучасні стандарти інтернету мають на увазі створення яскравих сторінок, що запам'ятовуються, тому веб-майстри все частіше використовують CSS, але мова HTML повністю не зникне, оскільки вона є основою багатьох інших систем.

### Структурні елементи web-сторінки

Отже, HTML – певна сукупність правил, за якими оформляється документ і показують, як буде він відображатися на web-сторінці. HTML – основний формат подання документів у мережі.

Зазвичай, основними структурними елементами більшості сторінок є:
*шапка* – вгорі веб-сторінки: назва сайту, головне меню, пошукова форма;

*підвал* – в самому низу веб-сторінки: авторські права, іконки соцмереж, форма підписки, тут часто дублюють навігацію;

*навігація* – головне меню сайту, посилання для переходу до інших розділів і підрозділів сайту;

*стаття* – основний текст з назвою, суть сторінки, супроводжувальні матеріали (картинки, таблиці, формули, ...), дата публікації, ім'я автора тощо;

*бічна панель* – часто вертикальна смуга праворуч або ліворуч від основного вмісту: посилання, рекламні блоки, форма голосування тощо.

Як правило, весь текст поділяють на розділи (підрозділи) – це основні структурні одиниці за розділення сайту на блоки.

Для формування сторінки використовують елементи, теги і їх атрибути (рис. 1.1).

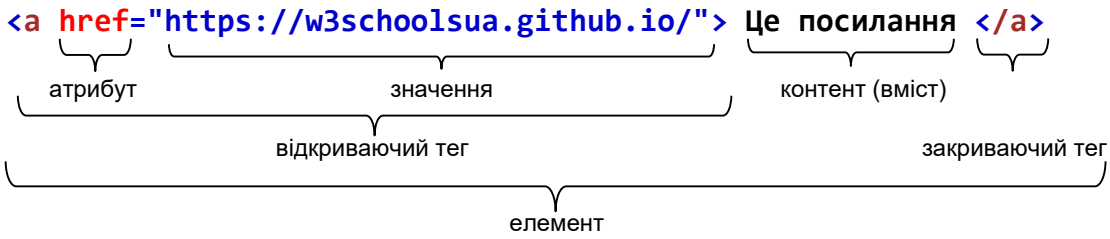



Рисунок 1.1 – Структура елементів web-сторінки

*Елемент* – це набір з тегів і їх вмісту. Сторінка складається з елементів.
*Теги* – команди, які виконує браузер.
*Атрибути* знаходяться всередині тегів, налаштовують теги.

**Основні правила написання тегів і атрибутів**

1. Відповідно до поточного стандарту HTML 5 теги можна писати як в нижньому, так і у верхньому регістрі:
   `<BODY>` , `<Body>`, `<BoDy>` – правильно,
   `< Body>` – неправильно (пробіл між відкриваючою кутовою дужкою і іменем тегу неприпустимий)
   *Рекомендовано: в нижньому регістрі <body>, <html> </p>*
2. Правильне написання атрибутів, значення яких не містять пробілу:
   `lang="en", lang='en'` – правильно;
   `lang=en` – правильно, можна без лапок (якщо немає пробілу!)
3. Якщо значення атрибута містить лапки, то значення береться в дужки.

**Види тегів**

Теги можуть бути:

– ***парними*** (є відкриваючий і відповідний закриваючий тег), наприклад,

```
<html> …. </html>
<head> …  </head>
<title> … </title>
<body> …  </body>
```

– ***непарними***, тобто, їх не треба закривати. Наприклад,

```
<!DOCTYPE html>
<meta>
```

Елементи web-сторінки можуть бути блоковими і рядковими.

***Блокові*** *елементи* становлять структуру сторінки і мають особливості:

– блоки розташовуються один під одним по вертикалі;
– не можна вставляти блоковий елемент всередину рядкового;
– займають весь допустимий простір по ширині;
– висота обчислюється автоматично, виходячи з вмісту.

Наприклад,

```
Абзаци:     <p>
заголовки: <h1> … <h6>
статті:     <article>
розділи:    <section>
довгі цитати: <blockquote>
списки марковані (з маркером): <ul>
списки нумеровані (з числами): <ol>
блоки загального призначення:  <div>
```

***Рядкові*** *елементи* використовують для форматування текстових фрагментів. Вони зазвичай містять одне або декілька слів і мають особливості:

– елементи, що йдуть попідряд, розташовуються на одному рядку і переносяться на іншу за необхідності;
– всередину допустимо вставляти текст або інші малі елементи, а от поміщати блокові елементи – заборонено.

Наприклад,

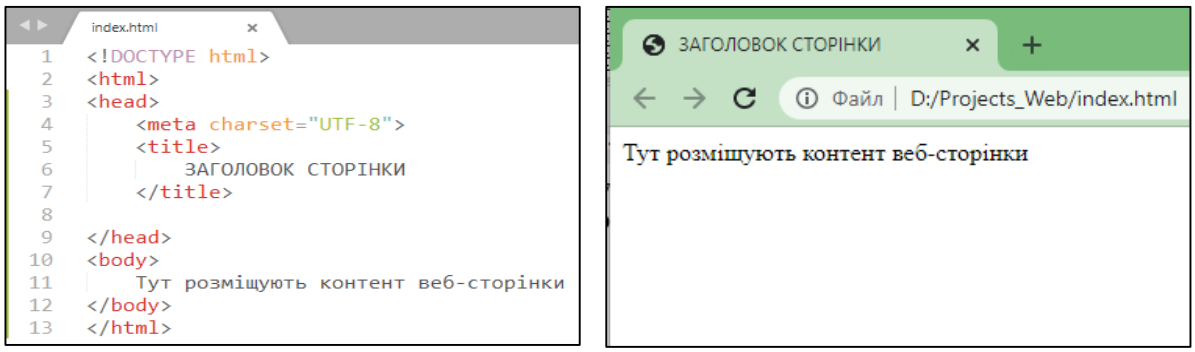
```
Посилання:      <a>
виділені слова: <em>
важливі слова:  <strong>
короткі цитати: <q>
абревіатури:    <abbr>
```

### Структура файлу для формування web-сторінки

Як правило файл з кодом, призначеним для формування сторінки, має містити такі основні контейнери (в які можуть входити інші контейнери):

**<!DOCTYPE html>** – сторінка написана за стандартом HTML5;
**<html>** … **</html>** – зовнішній контейнер, який містить заголовок сторінки і тіло сторінки;
**<head>** … **</head>** – контейнер, який містить інформацію про сторінку для браузера;
**<body>** … **</body>** – основний вміст (контент) web-сторінки.

Код і вигляд мінімальної сторінки може бути таким:




Для створення ***коментарів*** використовують такий елемент:

```
<!--  ... текст коментаря ... -->
```

### Основні теги мови HTML

### Тег <body> … </body>

Тег призначений для зберігання тіла веб-сторінки (його контенту), що відображається у вікні браузера.

Основними атрибутами тегу <*body*> є такі:

| Атрибут | Опис |
|---|---|
| **topmargin, bottommargin, leftmargin, rightmargin** | Відступ від верхнього, нижнього лівого та правого країв вікна браузера до контенту |
| **bgcolor,** | Колір фону веб-сторінки |
| **background** | Задає фоновий рисунок на веб-сторінці |
| **text** | Колір тексту в документі |
| **link, alink, vlink** | Встановлює колір посилань, колір активного та відвіданого посилання, відповідно |
| **bgproperties** | Визначає, чи буде прокручуватися фон разом з текстом |
| **scroll** | Встановлює чи відображати смуги прокручування |

Задавати ***кольори*** (фону, тексту, окремих елементів на сторінці) можна декількома способами:

– за допомогою імені кольору, наприклад:
  `red, green, magenda, yellow, …`
– за допомогою шістнадцяткового подання, наприклад:
  `#FF2316, #AABBCC, #FAB3F4, …`

### Оформлення тексту і документа

Існує ряд наперед визначених стилів для оформлення певних частин текстової інформації.

До них відносяться, наприклад, такі:

**<article>** – оформлення статті;

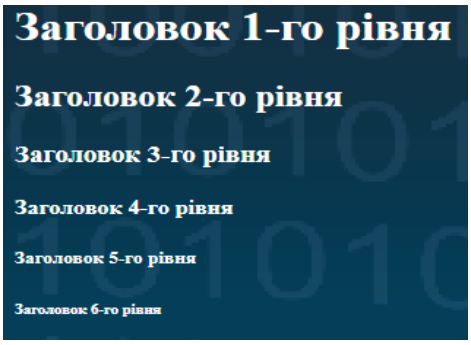
**<section>** – розділи;

**<blockquote>** – довгі цитати.

### *<H1> … <H6> – заголовки*

Передбачено шість видів наперед визначених заголовків, хоча їх стилі можна змінювати за власним уподобанням. Це парні теги, і виглядають вони, наприклад, так (фрагмент коду – зліва, вигляд – справа):

```
...
 <h1> Заголовок 1-го рівня </h1>
 <h2> Заголовок 2-го рівня </h2>
 <h3> Заголовок 3-го рівня </h3>
 <h4> Заголовок 4-го рівня </h4>
 <h5> Заголовок 5-го рівня </h5>
 <h6> Заголовок 6-го рівня </h6>
...
```

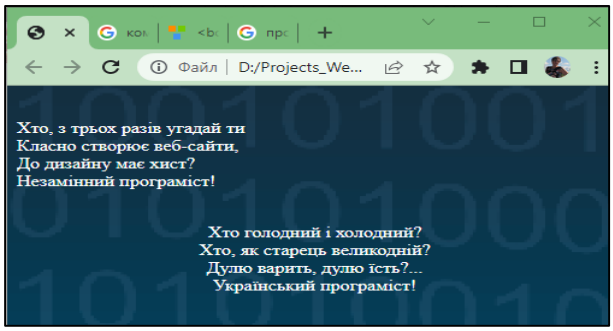



### * …  – параграф*

За використання тегу  відбувається перехід на новий рядок і здійснюється невеликий інтервальний відступ від попереднього абзацу. Цей тег має власний атрибут вирівнювання (align), але встановити інші атрибути можна буде за допомогою CSS.

Приклад.

```
<p>
    <br>Хто, з трьох разів угадай ти
    <br>Класно створює веб-сайти,
    <br>До дизайну має хист?
    <br>Незамінний програміст!
</p>
<p align="center">
    <br>Хто голодний і холодний?
    <br>Хто, як старець великодній?
    <br>Дулю варить, дулю їсть?...
    <br>Український програміст!
</p>
```




Для фізичного форматування тексту використовують такі теги:

**<br>** – перехід на новий рядок;
**** … **** – виділення жирним,
**** … **** – виділення курсивом,
**<u>** … **</u>** – виділення підкресленням,
**<s>** … **</s>** – виділення закресленням,
**<sub>** … **</sub>** – нижній індекс,
**<sup>** … **</sup>** – верхній індекс,
** ... ** – виділення важливих слів,
**<q> ... </q>** – короткі цитати,
**<abbr> ... </abbr>** – абревіатури.

Для кожного з цих тегів можна задати певний власний стиль.

### *Спеціальні символи HTML*

Існують безліч різних псевдолітер, які можуть бути відображені на сторінці, але які геть відсутні на будь-якій комп'ютерній клавіатурі. У

деяких випадках без них не можна обійтися. Використовувати їх дуже просто: у потрібному місці замість необхідної для відображення псевдолітери потрібно поставити наведений в таблиці символ або код, в кінці якого обов'язково ставиться крапка з комою!

| | |
|---|---|
| – | — |
| |   |
| § | § |
| © | © |
| « | « |
| » | » |
| ® | ® |
| ° | ° |

| | |
|---|---|
| ² | ² |
| ³ | ³ |
| ¼ | ¼ |
| ½ | ½ |
| ¾ | ¾ |
| × | × |
| ÷ | ÷ |

***<font> … </font>***

Цей тег застосовується, якщо потрібно змінити параметри шрифту деякого фрагменту тексту.

Атрибутами цього тегу є:

**`color`** – визначає колір тексту;

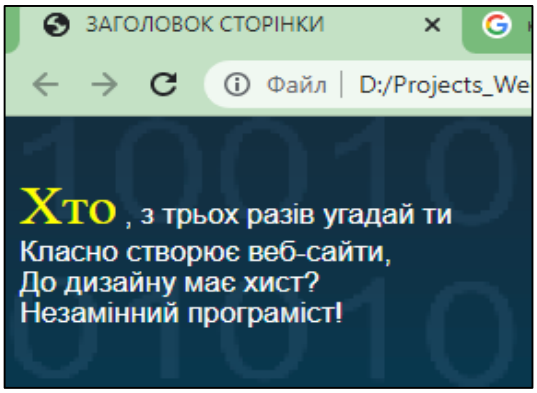
**`face`** – визначає гарнітуру шрифту,

**`size`** – визначає розмір шрифту.

Приклад.

```
<p>
    <br>
    <font color="Yellow" size="6">Хто</font>
    <font face="Arial">
        , з трьох разів угадай ти
        <br>Класно створює веб-сайти,
        <br>До дизайну має хист?
        <br>Незамінний програміст!
    </font>

</p>
```




**Зображення на сторінці**

Для цього використовуються два види тегів:

**`<img>`** – для відображення картинок формату gif, jpeg, png та інших на Web-сторінці;

**`<figure>`** – може містити зображення, відеоролик, схему, фрагмент коду, діаграму або навіть таблицю – майже все, що може з'явитися в потоці веб-контенту і має сприйматися як автономна одиниця.

***<img>***

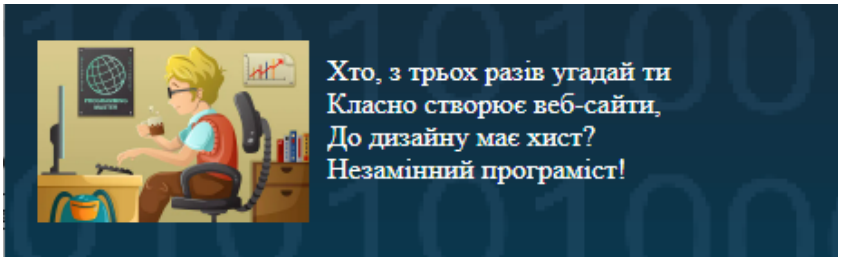
Тег <*img*> має ряд атрибутів:

| | |
|---|---|
| **src** | вказує шлях до зображення від файлу HTML |
| **alt** | альтернативний текст для випадку, коли картинка не завантажилась або є обмеження на обсяг завантаження |
| **align** | Визначає як рисунок буде обтікатися текстом |
| **border** | Товщина рамки навколо зображення |
| **width, height** | Ширина і висота зображення |
| **vspace, hspace** | Вертикальний та горизонтальний відступи від зображення до навколишнього контенту |

Наприклад, такий код:

```
<p>
<img  src="pr1.jpg" alt="Програміст зліва" align="left"
      width="150" height="100"  vspace="10" hspace="10" />
<br> Хто, з трьох разів угадай ти <br> Класно створює веб-сайти,
<br> До дизайну має хист?  <br> Незамінний програміст!
</p>
```

дасть такий вигляд сторінки з зображенням:



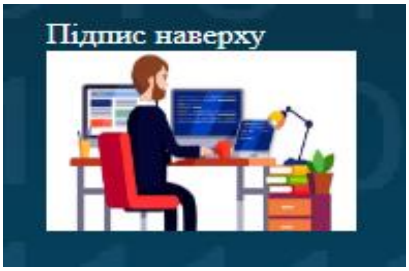

***<figure>***

Завдяки тегу <figcaption> ви можете вивести пояснення до вмісту, яке знаходиться всередині тегу <figure>. Тег <figcaption> має розміщуватися як перший або останній елемент всередині тегу <figure>.

Приклад. Додамо у код у html-файл:

```
<figure>
      <img src="pr1.jpg" width="150" height="100">
      <figcaption>Підпис внизу</figcaption>
  </figure>
      <figure>
      <figcaption>Підпис наверху</figcaption>
      <img src="pr2.jpeg" width="150" height="100">
   </figure>
```

Отримаємо:

## Посилання

Тег **<a>** створює посилання на іншу сторінку – зовнішнє посилання, або на певний елемент цієї сторінки – внутрішнє посилання.

### *Внутрішні посилання – якорі*

Внутрішні посилання зручні тоді, коли контент має великий обсяг і є необхідність доступу до різних частин документа.

У цьому випадку перед фрагментами, до яких потрібен швидкий доступ, потрібно проставити «якір», надавши йому ім'я:

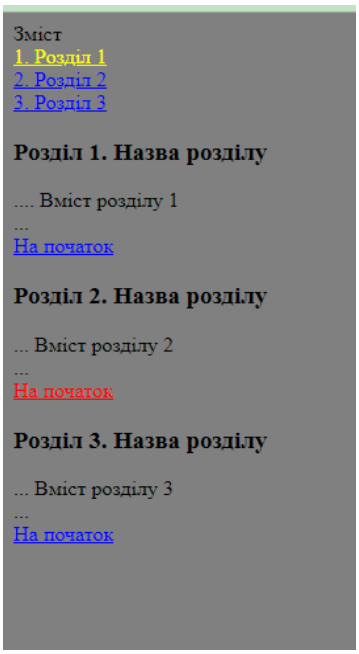
```
<a name="Part1"></a>
```

А в тих місцях, де потрібно здійснити швидкий перехід на потрібний фрагмент, необхідно поставити посилання (перед іменем якоря обов'язково має бути наявним символ '#':

```
<a href="#Part1">Частина 1</a>
```

Приклад (зліва – фрагмент коду, справа – результат).

```
<body bgcolor=gray link=Yellow vlink="blue"
alink="red">
      <a name = "Begin"></a>
      Зміст
      <br><a href="#part1"> 1. Розділ 1 </a>
      <br><a href="#part2"> 2. Розділ 2 </a>
      <br><a href="#part3"> 3. Розділ 3 </a>
      <a name="part1"></a>
      <p><h3> Розділ 1. Назва розділу </h3>
            .... Вміст розділу 1 <br>...
          <br> <a href="#Begin"> На початок </a>
      </p>
      <a name="part2"></a>
      <p><h3> Розділ 2. Назва розділу </h3>
            ... Вміст розділу 2      <br>...
            <br> <a href="#Begin"> На початок </a>
      </p>
      <a name="part3"></a>
      <p><h3> Розділ 3. Назва розділу </h3>
            ... Вміст розділу 3  <br>...
            <br> <a href="#Begin"> На початок </a>
      </p>
</body>
```




### *Зовнішні посилання*

Посилання на іншу сторінку цього самого сайту:

```
<a href="page2.html"> Посилання поруч </a>
```

Посилання на початкову сторінку зовнішнього сайту:

```
<a href="https://css.in.ua/"> Посилання на інший сайт </a>
```

Посилання на конкретну сторінку зовнішнього сайту:

```
<a href=https://css.in.ua/html/tag> Деякий текст </a>
```

Як посилання може бути деяке зображення:

```
<a href=https://google.com>
```

```
    <img src="google.png" width="80" height="50">
</a>
```

Тег <a> може мати ще й інші атрибути, крім основного атрибута *href*, який задає адресу документа, на який потрібно перейти.

Атрибут *target* вказує ім'я вікна або фрейма, куди браузер буде завантажувати документ. Якщо *target=”blank”*, то завантаження відбудеться у те саме вікно, звідки створено посилання.

**Списки**

В HTML передбачено три види списків:

- нумеровані списки;
- марковані списки;
- списки означень.

***<ol> … </ol> – нумерований список***

Теги *<ol> ... </ol>* визначають нумерований список, пункти якого визначаються тегами *<li> ... </li>*, причому тип нумерації задається атрибутом *type*, який може набувати таких значень:

**1** – нумерація арабськими цифрами,

**I, i** – нумерація римськими цифрами,

**A, a** – нумерація літерами (великими або малими).

Приклад.

```
<b><u>Засоби веб-програмування<u><b>
<ol type="A">
    <li>Мова розмірки гіпертексту HTML</li>
    <li>Каскадні таблиці стилів CSS</li>
    <li>Скриптова мова JavaScript</li>
    <li>Мова PHP</li>
</ol>
```

**Засоби веб-програмування**

A. Мова розмірки гіпертексту HTML
B. Каскадні таблиці стилів CSS
C. Скриптова мова JavaScript
D. Мова PHP

Як пункти меню можуть бути посилання (внутрішні і зовнішні), картинки, інші списки тощо. Вкладеність таких списків може бути досить глибокою.

***<ul> … </ul> – маркований список***

Марковані списки схожі до нумерованих, але в них немає чисел або літер, в них наявні деякі маркери, які також визначаються атрибутом *type*, і можуть набувати значень:

**disk** – крапка,

**circle** – маленьке коло,

**square** – зафарбований квадратик.

Приклад.

```
<b><u>Засоби веб-програмування</u></b>
<ul type="square">
    <li>Мова розмірки гіпертексту HTML</li>
    <li>Каскадні таблиці стилів CSS</li>
    <li>Скриптова мова JavaScript</li>
    <li>Мова PHP</li>
</ul>
```

**Засоби веб-програмування**

- Мова розмірки гіпертексту HTML
- Каскадні таблиці стилів CSS
- Скриптова мова JavaScript
- Мова PHP

Пунктами меню тут також можуть бути інші елементи.

Часто марковані і нумеровані списки використовують для створення меню, за допомогою якого можна здійснювати навігацію по сторінках сайту.

***<dl> … </dl> списки визначень***

Тег *<dl> (Definition List)* призначений для створення списку термінів (опису /визначень).

Кожен такий список починається з контейнера *<dl>*, куди може входити елемент *<dt>* (*Definition Term*)**,** що створює термін, або елемент *<dd> (Definition Description)*, який задає опис цього терміну.

Наприклад, наступний фрагмент коду демонструє використання списків означень:

```
<b><u>Проблеми у захисті програмного забезпечення</u></b>
<dl>
    <dt><i><b>промислове шпигунство</b></i>
       <dd>незаконне використання алгоритмів,
           що є інтелектуальною власністю
           автора, під час написання аналогів продукту
    <dt><i><b>крадіжка і копіювання</b></i>
        <dd>несанкціоноване використання ПЗ
    <dt><i><b>несанкціонована модифікація ПЗ</b></i>
        <dd>з метою впровадження програмних зловживань
    <dt><i><b>піратство </b></i>
        <dd>незаконне поширення і збут ПЗ.
</dl>
```

Як результат на сторінці побачимо таке:

**Проблеми у захисті програмного забезпечення**

***промислове шпигунство***
незаконне використання алгоритмів, що є інтелектуальною власністю автора, при написанні аналогів продукту

***крадіжка і копіювання***
несанкціоноване використання ПЗ

***несанкціонована модифікація ПЗ***
з метою впровадження програмних зловживань

***піратство***
незаконне поширення і збут ПЗ.

**Таблиці**

Для створення таблиці використовується парний тег

**<table> ... </table>**

який створює таблицю в HTML документі. Потрібно, щоб таблиця мала хоча б один рядок, що описується в тегах

**<tr> ... </tr>**

та одну комірку, описану в тегах

**<td> … </td>.**

Розподіл таблиці на верхній та нижній колонтитули, заголовок таблиці і тіло відбувається за допомогою таких тегів:

**<caption>** – заголовок таблиці,

**<thead> –** шапка таблиці,

**<tbody>** – тіло таблиці,

**<tfoot>** – нижній колонтитул таблиці (напис, що щось пояснює).

Теги *tbody, thead, tfoot* не обов'язкові, більшість сучасних браузерів підставляє їх самостійно, але їх можна використати для стилізації таблиці.

Крім того, можна керувати розташуванням, виглядом, вирівнюванням, форматуванням як таблиці загалом, так і її окремих рядків, комірок.

Для тегу <*table*> існує ряд атрибутів:

| | |
|---|---|
| **align** | визначає положення таблиці на екрані |
| **background, bgcolor** | визначає фоновий малюнок або фоновий колір в таблиці |
| **cols** | задає кількість стовпців у таблиці |
| **frame** | визначає тип рамки навколо таблиці |
| **border** | визначає товщину рамки, а також визначає, чи використовується таблиця для компонування макета сторінки (табличне верстання) |
| **width** | ширина таблиці |
| **cellpadding, cellspacing** | відступ від рамки до вмісту комірки та відстань між комірками, відповідно |

Крім того, в кожній з комірок можна задавати своє вирівнювання за допомогою атрибутів:

**align** – вирівнюваннння по горизонталі: **left, center, right**;

**valign** – вирівнювання по вертикалі: **top, middle, bottom.**

Приклад.

```
<table align="center" width="70%" bgcolor="white" border="1">
<caption><b>РОЗКЛАД ПАР В ВНТУ</b></caption>
<thead>
            <tr>  <th valign="center">Номер пари</th>
            <th valign="center">Час</th>
            </tr>
</thead>
<tr>        <td>8:00 – 16:00</td>
            <td>12:00 – 13:00</td>
</tr>
<tr>  <td>9:00 – 17:00</td>
      <td>13:00 – 14:00</td>
</tr>
```

```
<tr>  <td>11:00 – 19:00</td>
      <td>15:00 – 16:00</td>
</tr>
<tfoot>
      <tr>  <th colspan="2">
                  <i>Через кожні 45 хв. – перерва на 15 хв.</i>
            </th>
      </tr>
</tfoot>
</table>
```

Результат – на сторінці побачимо таку таблицю:

**РОЗКЛАД ПАР В ВНТУ**

| Номер пари | Час |
|---|---|
| 8:00 - 16:00 | 12:00 - 13:00 |
| 9:00 - 17:00 | 13:00 - 14:00 |
| 11:00 - 19:00 | 15:00 - 16:00 |
| ***Через кожні 45 хв. - перерва на 15 хв.*** | |

### Семантична верста сайту

До появи стандарту HTML5 вся розмітка сторінок здійснювалася переважно за рахунок елементів <*div*>, яким присвоювали класи (*class*) або ідентифікатори (*id*) для структурування і наочності розмітки. З їх допомогою в HTML-документі розміщували верхні і нижні колонтитули, бічні панелі, навігацію та багато іншого.

Стандарт HTML5 надав нові елементи для структурування, групування контенту і розмітки текстового вмісту. Нові семантичні елементи дозволили поліпшити структуру веб-сторінки, додавши смислове значення укладеному в них вмісту.

### Секційні елементи

**<header>** – парний тег (не є обов'язковим), який утворює вміст верхньої (шапки) частини сторінки або її секції. Об'єднує вступну інформацію і навігаційні елементи, може розташовуватися в будь-якій частині сторінки, наприклад:

```
<header>
      <h1> … </h1>
      <p> …  </p>
</header>
```

Особливості:

- у HTML-документі може міститися одночасно кілька елементів <*header*>;
- може містити основний заголовок, або групу заголовків, які також можна помістити в елементах <*hgroup*>;

– <*header*> не можна поміщати всередину елементів <*footer*>, <*address*> або іншого елемента <*header*>.

**<hgroup>** – використовується для групування елементів <*h1*> – <*h6*> в разі, коли заголовок має складну структуру, наприклад, має уточнювальні підзаголовки, альтернативні заголовки і т. д., наприклад:

```
<header>
      <hgroup>
            <h1> … </h1>
            <h2> … </h2>
      </hgroup>
</header>
```

**<nav>** – призначений для створення блока навігації веб-сторінки або всього веб-сайту, водночас не обов'язково має знаходитися усередині <*header*>. На сторінці може бути декілька елементів <*nav*>. Він не замінює теги <*ul*> або <*ol*>, він просто їх обрамляє.

Приклади:

| Використання <nav> для елементів списку | Параграфи, посилання всередині елемента <nav> | Заголовки всередині елемента <nav> |
|---|---|---|
| <nav><br><ul><br><li><a>...</a></li><br><li><a>...</a></li><br><li><a>...</a></li><br></ul><br></nav> | <nav><br> <a>...</a> <br> <a>...</a> <br></nav> | <nav><br><h2>...</h2><br><ul><br><li><a>...</a></li><br><li><a>...</a></li><br><li><a>...</a></li><br></ul><br></nav> |

**<article>** – використовується для групування записів – публікацій, статей. Являє собою незалежний, відокремлений блок, призначений для багаторазового використання. Як правило, починається з заголовка:

```
<article>
      <header>
          <h2>...</h2>
      </header>
      <p>...</p>
 </article>
```

**<section>** – є способом поділу сторінки або статті на тематичні розділи. Не використовується багаторазово, зазвичай містить заголовок. Не є блоком-обгорткою, для цих цілей доречніше використовувати елемент <*div*>. Наприклад,

```
<article>
      <h1>...</h1>
      <section>
           <h2>...</h2>
           <p>...</p>
      </section>
 </article>
```

**<aside>** – групує вміст, пов'язаний з навколишнім контентом безпосередньо, але яке можна вважати окремим. Найчастіше елемент позиціонується як бічна колонка (як у книгах) і містить в собі групи елементів <nav>, цифрові дані і деякі потрібні цитати:

```
<aside>
        <h2>...</h2>
        <p>...</p>
</aside>
```

**<footer>** – формує вміст нижньої частини (підвалу) сторінки або її секції. Призначений для розміщення деякої інформації про веб-ресурс, наприклад, відомості про авторські права, посилання на умови використання, контактну інформацію, посилання на пов'язаний вміст і т. д. В одному веб-документі може бути декілька елементів <footer>. Як кожна сторінка, так і кожна стаття може мати свій елемент <footer>.

```
<footer>
        <address>...</address>
        <small>...</small>
</footer>
```

**<address>** – використовується для визначення контактної інформації автора/власника документа або статті. Для позначення автора документа тег розміщують всередині елемента <*body*>, для відображення автора статті – всередині тега <*article*>. У браузері зазвичай відображається курсивом.

### Групування контенту

**<main>** – групує основний контент веб-сторінки. Вміст елемента має бути унікальним на сторінці і не має відображатися де-небудь ще на сайті. Тому що повторюваний на сайті контент, наприклад навігація по сайту, пошук по сайту, логотип, контактні дані та інше, не можна поміщати в елемент <main>.

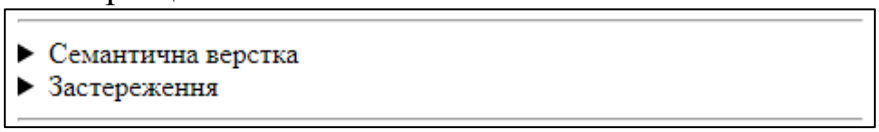
**<figure>** і **<figcaption>**. Про цей елемент вже йшлося вище. Він являє собою автономний контент, можливо з заголовком <*figcaption*>, який є самостійним елементом потоку. Елемент може бути переміщений з основного контенту документа в бічну колонку або додаток, не зачіпаючи потік документа. За допомогою елемента <*figure*> можна додавати замітки до ілюстрацій, фотографій, діаграм, фрагментів коду, даючи підписи або коментарі.

**<details>** і **<summary>** – використовується як низхідне меню або додаткові деталі, які користувач може приховати або показати. Тут <*summary*> використовується як видимий заголовок елемента <*details*>, за натискання на який можна показати/приховати його.

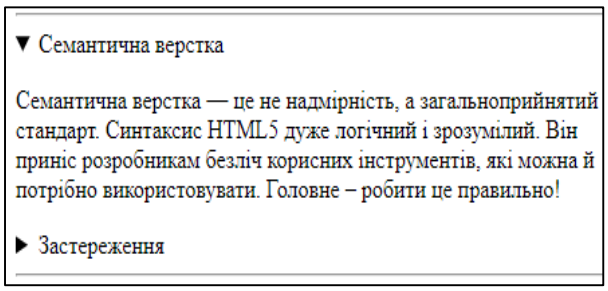
Наприклад, наступний фрагмент коду демонструє виведення на сторінку двох невеликих повідомлень, які виглядають як список:

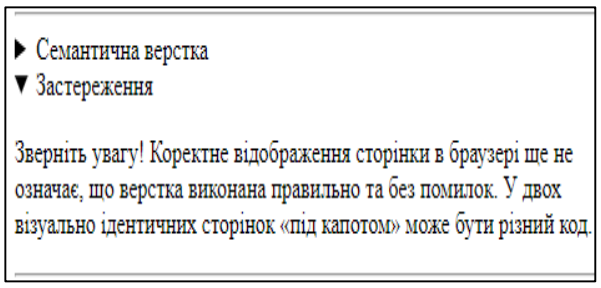
```
<hr>
  <details>
    <summary> Семантична верстка </summary>
      <p>Семантична верстка — це не надмірність, а загальноприйнятий
         стандарт. Синтаксис HTML5 дуже логічний і зрозумілий. Він
         надає розробникам безліч корисних інструментів, які можна й
         потрібно використовувати. Головне – робити це правильно!
      </p>
  </details>
  <details>
    <summary> Застереження </summary>
    <p>Зверніть увагу! Коректне відображення сторінки в браузері ще не
       означає, що верстка виконана правильно та без помилок. У двох
       візуально ідентичних сторінок «під капотом» може бути різний код.
    </p>
  </details>
<hr>
```

На сторінці маємо:




А при натисканні на трикутничок зліва від тексту з'являється підготовлене повідомлення:




### Семантичні елементи для текстового вмісту

**<time>** – визначає час (24 години) або дату за григоріанським календарем з можливим зазначенням часу і зміщення часового поясу. Текст, укладений в цей тег, не має стильового оформлення браузером. Для цього тегу доступний атрибут *datetime*, вміст якого вказує на те, що буде бачити користувач на екрані свого комп'ютера.

Наприклад, внаслідок виконання фрагменту коду:

```
<h3>План навчання</h3>
    <time>01.09.2022</time>  Перший семестр
<br><time>01.01.2023</time>  Зимові канікули
<br><time>01.02.2023</time>  Другий семестр
<br><time>27.06.2023</time>  Літні канікули
```

на сторінці побачимо таке:

**<mark>**. Текст, поміщений всередину тегу <mark>, виділяється, за замовчуванням, жовтим кольором (можна змінити, задавши css-стилі). За допомогою цього тегу можна відзначати важливий вміст, а також деякі ключові слова. Наприклад, якщо використати такий фрагмент коду:

```
<p>ВАЖЛИВО! <mark> Не всі набори посилань потрібно обертати
   тегом <i>nav</i>.</mark> Якщо такі елементи як, наприклад
   <i>footer</i> і <i>header</i> справляються з покладеними на
   них завданнями в одиничному екземплярі, то <i>nav</i> можна
   використовувати більше одного разу на сторінку.
</p>
```

то результат може виглядати так:

ВАЖЛИВО! Не всі набори посилань потрібно обертати тегом *nav*. Якщо такі елементи як, наприклад *footer* і *header* справляються з покладеними на них завданнями в одиничному екземплярі, то *nav* можна використовувати більше одного разу на сторінку.

**<wbr>** – показує браузеру місце, де можна додати розрив довжини рядка в разі потреби.

**Висновок щодо семантичної верстки**

HTML5 для семантичної верстки сайтів пропонує нові семантичні елементи: <*article*>, <*aside*>, <*details*>, <*figcaption*>, <*figure*>, <*footer*>, <*header*>, <*main*>, <*mark*>, <*nav*>, <*section*>, <*summary*>, <*time*>.

Звичайно, можна для зручного і гарного вигляду використовувати інші елементи, надаючи їм певні стилі за допомогою CSS, але в такому разі не можна отримати 100 % семантично валідної верстки.

Приклад правильної семантичної розмітки HTML5:

*Загальна розмітка сторінки*

```
<html>
   <head>
        <title>...</title>
   </head>
   <body>
        <header>
            ...
        </header>
        <nav>
            ...
```

*Основний контент*

```
<main>
        <h1>...</h1>
        <p>...</p>
        <section>
          <h2>...</h2>
          <p>...</p>
        </section>
        <section>
           <h2>...</h2>
           <p>...</p>
```

**План навчання**

01.09.2022 Перший семестр
01.01.2023 Зимові канікули
01.02.2023 Другий семестр
27.06.2023 Літні канікули

```
        </nav>
        <main>
            ...
        </main>
        <footer>
            ...
        </footer>
    </body>
</html>
```

```
            </section>
            <section>
                <h2>...</h2>
                <p>...</p>
            </section>
            ...
        </main>
```

**Контрольні запитання**

1. Що таке HTML? Чим вона відрізняється від інших мов програмування?
2. Назвіть основні етапи становлення мови HTML.
3. Які структурні елементи сторінок Ви можете назвати? Чи обов'язкова присутність усіх цих складових? Чому?
4. Які види тегів Ви знаєте? Наведіть приклади.
5. Що таке елемент коду сторінки, тег, атрибути тегів?
6. Які основні правила написання елементів тегів і атрибутів?
7. Які основні складові коду сторінки?
8. Як відформатувати основні параметри тіла сторінки?
9. Яким чини встановити кольори посилань?
10. Як можна вставити зображення на сторінку і яким чином керувати параметрами зображення?
11. Яким чином можна керувати форматуванням тексту?
12. Що собою являють зовнішні посилання? Яким чином вони вставляються у сторінку?
13. Що таке внутрішні посилання? Для чого вони потрібні? Як реалізувати внутрішні посилання?
14. Як вставити в сторінку списки? Які види списків Ви знаєте? Як їх реалізувати?
15. Що таке списки означень? Для чого їх використовують і як реалізують?
16. Як вставити таблицю в сторінку? Які складові може мати таблиця?
17. Яким чином можна керувати параметрами таблиці? Назвіть основні атрибути для керування виглядом таблиць.
18. Що таке семантична верстка сайту?
19. Які додаткові теги використовують для верстки сайтів?

**Практична робота № 1**

***Мета роботи***

- Ознайомитись з основними тегами мови HTML.
- Навчитись створювати найпростіші web-сторінки.
- Засвоїти засоби для керування розташуванням інформації та її форматуванням на сторінках.

***Порядок виконання роботи***

1. Обрати тему сайту з переліку індивідуальних завдань для виконання лабораторних робіт.
2. Скориставшись пошуком в мережі Інтернет, підібрати текстові і графічні матеріали (історичні відомості, правила гри або описи алгоритмів, фотографії, зображення тощо – все те, що можна розповісти з теми індивідуального завдання).
3. Продумати структуру майбутнього web-сайту (сайт має містити не менше двох сторінок) і, здійснивши розмітку сторінок, розмістити інформаційний матеріал.
4. Вимоги до web-сайту:
   - під час створення сторінки користуватись ***виключно HTML*** (наразі обходитись *без CSS та JS* – це буде використано у наступних лабораторних роботах);
   - сторінка обов'язково має містити ***таблиці***;
   - знайти декілька ***посилань*** на відповідну тему і ввести їх у контент сторінки;
   - розробити невелике ***меню*** і реалізувати ***навігацію*** по сайту і по сторінках з текстом (де це необхідно);
   - сторінка має містити ***списки***;
   - розташування ***картинок*** на сторінці має бути різноманітним (по центру, біля лівого/правого краю, в тексті, з підписом і без підпису тощо…).

   Рекомендації:
   - використовувати ***семантичну верстку*** сторінок;
   - кожну структурну одиницю помістити в ***окремий блок*** (для зручності подальшої модифікації);
   - у ***підвалі*** сторінки розмістити інформацію про автора.

***Варіанти індивідуальних завдань (теми сайтів)***

| № | Тема |
|---|---|
| 1 | Методи шифрування  на основі шифру Цезаря |
| 2 | Методи шифрування за допомогою магічних квадратів, методом подвійної перестановки та методом подвійного квадрата Уітстона |
| 3 | Шифрування методами Атбаш та Альбам |
| 4 | Шифрування методом Віжинера |
| 5 | Шифр Енігми |
| 6 | Капча |
| 7 | Паролі |
| 8 | Піктограмопаролі |
| 9 | Смайлики |
| 10 | Хмарні технології |
| 11 | Цифри |

| | |
|---|---|
| 12 | Сучасна криптографія |
| 13 | Формати графічних файлів |
| 14 | Формати музичних файлів |
| 15 | Історія комп’ютера |
| 16 | Хакери |
| 17 | Топ вірусів |
| 18 | Тор найкращих хакерів |
| 19 | Топ крутих зломів |
| 20 | Мови програмування |
| 21 | Годинники |
| 22 | Етикет |
| 23 | Ігрові програми для розвитку пам’яті |
| 24 | Гороскопи |
| 25 | Блокчейн |
| 26 | Контроль варіантів і тем лабораторних робіт |
| 27 | Шифрування азбукою Морзе |
| 28 | Аналіз ігрових програм |
| 29 | Музичний плейєр |
| 30 | Сім Чудес України |

**Додаткові джерела**


1. Бородкіна І. Л., Бородкін Г. О. Web-технології та Web-дизайн: застосування мови HTML для створення електронних ресурсів : навч. посіб. / І. Л. Бородкіна, Г. О. Бородкін. Київ: Ліра-К, 2020. 212 с.
2. W3schoolsUA. українською. HTML Підручник [Електронний ресурс] : URL: https://w3schoolsua.github.io/html/index.html#gsc.tab=0.
3. Підручники HTML і CSS. [Електронний ресурс]: URL: https://htmlbook.at.ua/.
4. HTML5 Семантична верстка. [Електронний ресурс]: URL: https://avivi.pro/ua/blog/html5-semanticheskaya-vyerstka/.

# 2 ОСНОВИ CSS. ВИКОРИСТАННЯ CSS ДЛЯ СТИЛІЗАЦІЇ WEB-СТОРІНОК

## 2.1 Короткі історичні відомості

У 1990-х виникла потреба стандартизувати web-інструменти, створити загальні правила, за допомогою яких програмісти та web-дизайнери могли б створювати сайти. Як результат цієї необхідності з'явилися мови HTML 4.01 і XHTML, а також стандарт CSS.

Перша згадка про CSS була у 1994 році, коли норвежець Хоком Віум Лі запропонував використовувати CSS (*Cascading Style Sheets,* каскадні таблиці стилів) для стилістичного оформлення веб-сторінок. Йому не одразу вдалося просунути свою технологію, – тільки через декілька років йому вдалося привернути увагу до CSS. Отже, 17 грудня 1996 року опубліковано першу специфікацію (CSS1) і вона була рекомендована до використання Консорціумом Всесвітньої павутини (W3C).

Після невеликого успіху стан справ у технології CSS набагато покращився, і у травні 1998 р. (через 2 роки) було прийнято рекомендацію W3C для CSS2. Наступним етапом була CSS 2.1 – версія W3C від вересня 2009 року, побудована на базі CSS2 і була роботою над виправленням існуючих помилок. Сьогодні актуальною є версія CSS3, яка максимально розширена порівняно з попередніми версіями. CSS3 містить багато новинок, як то: ефекти тіней, закруглені кути у блоків, з'явилася можливість встановлювати зображення як фон і використовувати їх як межі та багато іншого. Набагато простіше й зручніше стала робота з анімацією –для її створення достатньо можливостей самої CSS3, можна навіть не використовувати мову JavaScript.

Специфікація CSS4 розробляється ще з 2011 року. Модулі CSS4 побудовані на основі CSS3 і доповнюють їх новими властивостями та значеннями. Всі вони існують поки що як чернетки (working draft) і на цей час офіційно не затверджені.

Сьогодні CSS – це загальноприйнятий стандарт розробки, який приймається всіма без винятку компаніями-розробниками, що показує його значущість і необхідність.

## 2.2 Основні поняття CSS

Різниця між CSS і HTML полягає у тому, що:

– HTML використовується для структурування вмісту сторінки;
– CSS використовується для форматування цього структурованого вмісту.

Іншими словами, CSS – це мова, яка описує стиль HTML документа, тобто, як мають відображатися HTML елементи.

До переваг CSS відносять таке:

1. CSS дозволяє значно *скоротити розмір коду* і зробити його читабельним.
2. CSS дозволяє *задавати особливі параметри*, які не можна задати тільки мовою HTML. Наприклад, прибрати підкреслення у посилань.
3. CSS дозволяє *легко змінювати зовнішній вигляд* сторінок.
4. CSS дозволяє здійснювати *блокову верстку* сайту. CSS (каскадна або блокова верстка) прийшла на заміну табличній верстці веб-сторінок. Головна перевага блокової верстки – розділення вмісту сторінки (даних) та її візуальної презентації (оформлення).

Основними поняттям CSS є поняття стилю.

***Стиль*** – це набір параметрів, що задає зовнішнє подання об'єкта.

***Таблицею стилів*** називають набір стилів усіх елементів.

Якщо для одного елемента задано декілька стилів (каскад стилів), то застосовується ***каскадування***, яке визначає пріоритет того чи іншого стилю.

Базовий синтаксис опису окремого стилю CSS такий:


Наприклад,

```
p { color: #333333; text-align: justify; }
body { background-color: #FF0000; }
H2 { font-size: 48pt; font-family: Arial; }
```

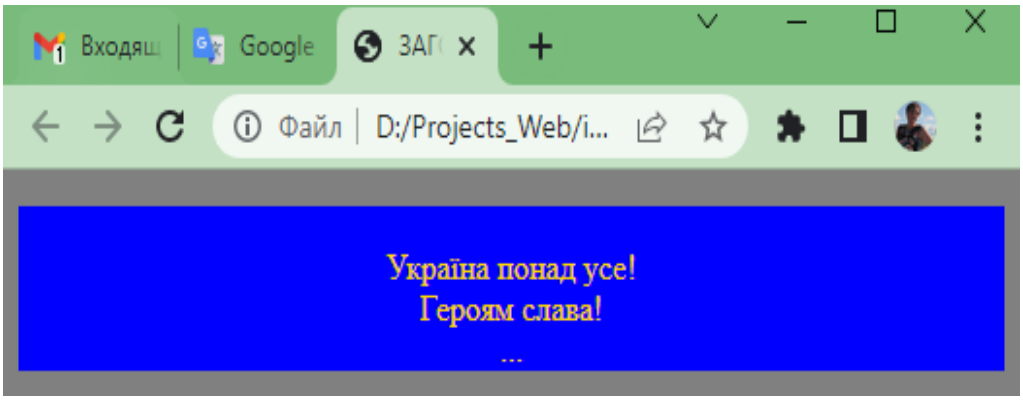
### 2.3 Підключення CSS до HTML-документа

Є декілька способів підключення стилів до html-коду.

1. ***Перший спосіб*** полягає у підключенні стилю «на льоту»: у тому місці, де треба застосувати деяке форматування, використовують атрибут *style* поточного тегу. Наприклад, у наступному фрагменті коду зміна стилю буде діяти лише на цей абзац.

```
<body bgcolor=gray link=Yellow vlink="blue" alink="red">
   <p style="background-color:blue;
             text-align: center; color: gold;" >
      <br>Україна понад усе!
      <br> Героям слава!<br>...
   </p>
...
```

Результат у вікні браузера буде такий:




2. ***Другий спосіб*** – використати внутрішній тег *<style> ... </style>*, тобто прописати потрібні стилі всередині вказаних тегів. Наприклад, наступний фрагмент коду виконує ті самі дії, але потрібно зауважити, що у цьому випадку всі абзаци виділені за допомогою тегів * ... * будуть відображені саме цим стилем.

```
<head>
    <meta charset="UTF-8">
    <title> ЗАГОЛОВОК СТОРІНКИ </title>
    <style type="text/css">
        p { background-color:blue;
            text-align: center;
            color: gold;
        }
    </style>
</head>
<body bgcolor=gray link=Yellow vlink="blue" alink="red">
    <p>
        <br>Україна понад усе! <br> Героям слава!<br>...
    </p>
...
```

3. ***Третій спосіб*** – використати зовнішній файл (з розширенням *.css*), що містить перелік усіх стилів, підключивши його за допомогою тегу *<link>*. Наприклад,

```
<html>
<head><title> ЗАГОЛОВОК СТОРІНКИ </title>
    <meta charset="UTF-8">
    <link   rel="stylesheet"
            type="text/css"
            href="Style/style.css"/>
</head>
...
```

Це досить зручно, оскільки один і той самий файл стилів можна використовувати для різних сторінок. Тобто, можна, не змінюючи жодного тегу в html-файлі, повністю змінити вигляд сторінки, внісши зміни лише у css-файл.

### 2.4 Види селекторів

**1.** ***Селектор за елементом (за тегом)*** **–** дозволяє відібрати у веб-документі всі однойменні дескриптори. Наприклад, змінити колір усіх абзаців або задати фон і колір усіх заголовків:

```
p { color: #F00; }
h1{ font-size: 0.9rem;  }
```

**2.** ***Селектор за класом.*** Створює клас, який буде використовуватись у разі виклику його в дескрипторі. Ім'я селектора починається з крапки. Наприклад, опишемо в таблиці стилів селектор:

```
.card {
      background: # 333;
}
```

Тоді у файлі .html його використовуємо так:

```
<Div  class = "card" >
       Тут якась інформація
</Div>
```

**3.** ***Селектор-ідентифікатор***. Цей селектор застосовується за потреби виділити один елемент, унікальний до інших елементів в документі. Ім'я селектора починається з символу решітки (#). Наприклад, у css-файлі:

```
#nodecor {
   text-decoration: underline;
}
```

Тоді у html-файлі використовуємо його так:

```
<a href=http://Google.com" id="nodecor"> Сторінка Google</a>
```

До простих селекторів відносять ще такі:

- селектори за атрибутом;
- контекстний селектор;
- селектор дочірніх елементів;
- селектор псевдокласів;
- селектори на будь-який елемент та інші.

Інформацію щодо їх використання легко знайти в інформаційних джерелах.

**4.** ***Групові селектори.*** В імені селектора використовується більше одного імені дескриптора. Наприклад весь текст на сторінці має бути одного кольору. Тоді виберемо всі абзаци і назви розділів:

```
p, h1{ color:#вCC0000 ; }
```

Можна групувати і селектори-класи:

```
div, .card, .cardFirstr { background: rgb(243, 243, 109); }
```

**5.** ***Елемент з класом.*** Можна стилізувати конкретний елемент, якщо у нього є певний клас. Наприклад,

```
p.example {  …   }
```

Цей селектор вибере всі *p*, у яких є клас *example*.

Або, наприклад, задано селектор:

```
.main.active { … }
```

Селектор вибере всі елементи з класом *main*, у яких є і клас *active*:

```
<div class = "main active">
  Деяка інформація
</div>
```

### 6. *Вкладені селектори*

Вкладеність на будь-якому рівні:

```
.page  p { color: red;   }
```

Цей селектор застосує червоний колір до всіх <*p*>, які знаходяться всередині елемента з класом *page* на будь-якому рівні вкладеності.

Для того, щоб розуміти складені селектор, потрібно читати їх справа наліво. Наприклад,

```
.page > .part { … }
```

Цей селектор вибере всі елементи *.part*, які знаходяться на першому рівні вкладеності в *.page*.

Або, якщо задано селектор:

```
.main .side-menu .menu-item { … }
```

він застосується до всіх елементів *.menu-item*, які знаходяться всередині елементів *.side-menu*, що знаходяться всередині елементів *.main*.

## 2.5 Основні параметри селекторів

Кожен з селекторів для опису стилів елемента має певні параметри, притаманні саме цьому селектору, які можна бачити у вигляді списку властивостей у будь-якому середовищі для розробки web-сторінок.

### 2.5.1 Текст і шрифт

Для кожного з елементів можна встановлювати значення параметрів шрифту і тексту:

**font-weight:** [bold|normal|number] – жирність шрифту;

**font-style:** [normal|italic|oblique] – нахил шрифту;

**font-size:** number – розмір шрифту;

**font-family:** name – гарнітура шрифту ;

**color:**number – колір шрифту;

**bgcolor:** number – колір фону;

**background:** url – текстурний фон;

**text-align:** [left|right|center|justify] – вирівнювання;

**text-indent:** number – відступ червоного рядка;

**ght:** number – відступ від границі справа.

Приклади:

```
body { color:#000000; background-color:#FFCC66; }
H1 {color:#ffffff; background-color:#000000; }
H2 {color:rgb(0,0,0); background-color:rgb(225,225,225);}
p{  color: rgb(39, 7, 7);
    background-color: rgb(105, 105, 146);
    text-align: justify;
 }
.dropbtn {
    background-color: #4CAF50;
    color: white;
    font-size: 16px;
    }
```

### 2.5.2 Сімейства шрифтів

Якщо встановлено такий селектор:

```
p { font-family: 'Roboto'; }
```

а на пристрої користувача немає цього шрифту, то він не побачить тексту. У цьому випадку спрацьовує ***fallback*** (фолбек):

```
p { font-family: 'Roboto', 'Helvetica Nue', sans-serif; }
```

В кінці списку фолбек-шрифтів завжди вказується сімейство, до якого входять попередні шрифти.

Існує декілька сімейств шрифтів:

| *serif* | *sans-serif* | *monospace* | *cursive* | *fantasy* |
|---|---|---|---|---|
| Bra | Ad | Consolas | Bickham Script Pro |  |
| шрифти с засічками або *антиква (*Times New Roman, Verdana, ...) | шрифти без засічок, *гротеск* або *рублені шрифти (*Arial, Lucida Consoles, ...) | моноширинні шрифти | рукописні шрифти | алегоричні шрифти – які не потрапляють в інші семейства |

### 2.5.3 Способи задання кольорів

Колір тексту, фону, рамки і інших елементів можна задавати декількома способами.

**Ім'я кольору**: "red", "green", "blue", "lemonchiffon".
**HEX-формат**: #3ae6ca, #FD6AE8.
**RGB** (red, green, blue): rbg (255, 123, 13).
**RGBA** = RGB + прозорість: rgba (144, 32, 16, 0.5).
**HSL –** колірна модель (тон, насиченість і світлість): hsl (120, 100%, 50%).
**HSLA**=- HSL + прозорість.

### 2.5.4 Фонове зображення (`background`)

**`background-image:`** `url` – для вставки фонового зображення;

**`background-repeat:`** `[repeat, repeat-x, repeat-y, n-repeat]` – повторення фонового зображення;

**`background-attachment:`** `[scroll, fixed]` – визначає, чи фіксується фоновий рисунок, чи прокручується разом з вмістом сторінки;

**`background-position:`** `[top right, number, …]` – розташування фонового рисунка.

## 2.6 Групування елементів (*`span`* і *`div`*)

Часто разом з атрибутами *class* та *id* використовується групування для структурування документа.

1. ***Групування за допомогою *** – використовується в елементах рівня одного блока. <*span*> – нейтральний елемент, який нічого не додає до вмісту документа, використовується для візуальних ефектів стосовно окремих елементів.
   Наприклад, встановивши селектор

```
.wow { color: red; }
```

і застосуватий цей стиль у коді для виділення слова «хакери»:

```
Якщо ми запитаємо дітей «Хто такі <span class="wow">хакери</span>», то майже
У наш час, <span class="wow">хакерами</span> називають тих, хто виявляє слабк
```

То отримаємо такий візуальний ефект:

Якщо ми запитаємо дітей «Хто такі хакери», то майже кожна дитина відповість, що це «молодий дядько, весь у чорному, з капюшоном на голові і який сидить перед комп'ютером вночі». А потім ще подумає і додасть: «І з колготками на голові!». У наш час, хакерами називають тих, хто виявляє слабкі місця в комп'ютерних

2. ***Групування за допомогою *** – застосовується для групування одного або більше блок-елементів.

## 2.7 Тип відображення елемента (*`display`*)

**`display:none;`** – елемент перестає відображатися на сторінці;

**`display:block;`** – блоковий елемент. Можна задати ширину, висоту, межі, відступи;

**`display:inline;`** – рядковий елемент. Задання ширини і висоти не впливає на *inline*-елементи. Задання границь і відступів буде змінювати положення навколишнього тексту, але не впливатиме на стан навколишніх блокових елементів;

**`display:inline-block;`** – середнє між блоковим і рядковим елементом. Можна задати ширину, висоту, межі та відступи, але він не буде

створювати перенесення рядка до і після себе, на відміну від блокових елементів. За допомогою цього типу можна розташовувати блоки горизонтально в ряд;

**`flex`** *і* **`inline-flex`** – це флексбокси. Елементи всередині них розташовуються за певними правилами, але зовні вони поводяться, як блоки і інлайн-блоки, відповідно. Про технологію *flexbox* будемо говорити далі.

**Контрольні запитання**

1. Що Ви знаєте з історії виникнення і розвитку CSS?
2. Що таке стиль?
3. Чому стилі назвали каскадними?
4. Що таке таблиці каскадних стилів?
5. Як розшифровується абревіатура CSS?
6. Що означає такий запис: *p {font-size: 40pt; color: green; font-family: "Comic Sans MS"}* ?
7. Що означають такі параметри форматування*: background, font-family, font-size, font-weight, color, text-decoration, text-align, line-height* ?
8. Назвіть три способи підключення стилів із HTML-документом.
9. Що таке зовнішня таблиця стилів ?
10. Як підключають зовнішню таблицю стилів?
11. Як розміщують внутрішню таблицю стилів?
12. З яких елементів складається стиль?
13. Які види селекторів бувають?
14. У яких випадках використовують селектори за ідентифікатором?
15. Як описують стиль для селекторів за класом і за тегом?
16. Наведіть приклади опису селекторів за класом, за тегом, за ідентифікатором.
17. Що таке групові селектори? Як їх описують?
18. Коли використовують вкладені селектори? Як їх треба читати?
19. Як записують коментарі в CSS?
20. Опишіть основні властивості роботи з текстом.
21. Що таке блок та якими тегами його описують?
22. У чому полягає різниця між атрибутами *id* і *class* тегу *di* ?
23. Які одиниці вимірювання CSS для визначення полів і відступів можна використовувати?
24. Яким чином можна задати кольори елементів?
25. Які сімейства шрифтів Ви знаєте і як їх встановлювати?
26. Як задати фонове зображення і його властивості (прокручування, розмір, повторення тощо)?
27. Які теги призначені для групування елементів? Яка різниця між ними?
28. Для чого використовують властивість *display*? Якого значення може набувати ця властивість?

## Практична робота № 2

### *Мета роботи*

- Ознайомитись з основними можливостями CSS.
- Навчитись застосовувати форматування у web-сторінках.
- Засвоїти на практиці різні способи використання селекторів CSS.

### *Завдання і порядок виконання роботи*

1. Удосконалити розроблений web-сайт, використовуючи форматування за допомогою CSS.
2. Обов'язково використати і продемонструвати під час захисту *три види* вбудовування стилів CSS.
3. Підготувати *не менше 2-х таблиць* каскадних стилів, за допомогою яких web-сторінка виглядатиме кардинально інакше:
   - різне форматування тексту,
   - різне оформлення контенту сторінок;
   - різне розташування елементів web-сторінок тощо.

### Додаткові джерела


1. W3schoolsUA. українською. CSS : підручник. [Електронний ресурс]: URL: https://w3schoolsua.github.io/css/index.html#gsc.tab=0.
2. Довідник CSS атрибутів. [Електронний ресурс]: URL: https://html-css.co.ua/dovidnuk-css-atrubytiv/.
3. Український веб-довідник. [Електронний ресурс]: URL: https://css.in.ua/.
4. METANIT.COM. [Електронний ресурс]: URL: https://metanit.com/web/html5/.
5. Безкоштовний довідник по CSS. [Електронний ресурс]: URL: https://cssreference.io.

# 3 БОКСОВА МОДЕЛЬ ВЕРСТАННЯ WEB-СТОРІНОК

## 3.1 Поняття боксової (блокової) моделі

В CSS термін «блокова модель» використовується, коли говорять про дизайн і верстку. Боксова модель описує бокси, що генеруються для HTML-елементів.

Більшість елементів HTML можна розглядати як блоки (коробки). Кожен блок (бокс) має детальні опції для визначення полів, рамок, заповнення та вмісту кожного елемента (рис. 3.1).

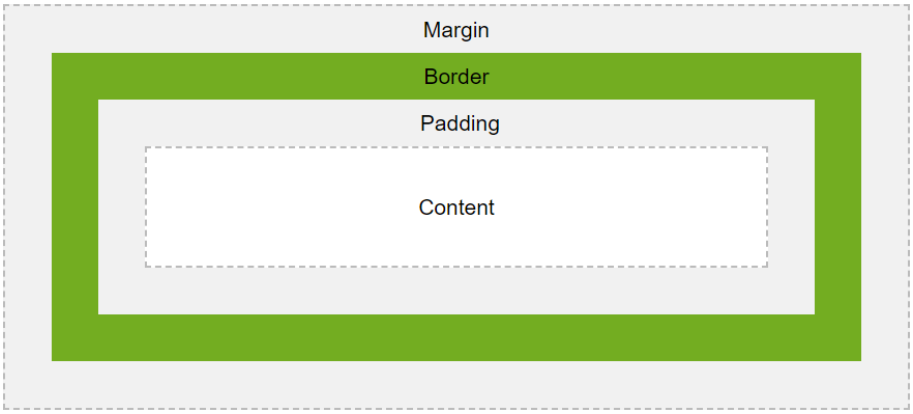



Рисунок 3.1 – Схема розмітки параметрів блока

Основними областями самих боксів є такі:

**`margin`** – зовнішній відступ;

**`border`** – межа між внутрішнім і зовнішнім;

**`padding`** – внутрішній відступ, під цим відступом починається фон елемента;

**`content`** – вміст елемента.

Кожний з параметрів блока може мати декілька підпараметрів. Так, параметр *margin* має чотири підпараметри, які перераховують через дефіс після назви параметра або після двокрапки у скороченому поданні. Наприклад,

```
body { margin-top: 100px;
       margin-right: 40px;
       margin-bottom: 10px;
       margin-left: 70px;
}
```

або скорочена форма:

```
body { margin:100px 40px 10px 70px;}
```

Оскільки параметрів стилів дуже багато, і кожний з параметрів має велику кількість підпараметрів, то знати їх всі і пам’ятати не має необхідності. Для того і створені програмні середовища, щоб допомогти у цій нелегкій, але цікавій справі.

### 3.2 Керування областями блокової моделі

**`margin, padding`**

Параметри зовнішніх (*margin*) та внутрішніх (*padding*) відступів мають підпараметри: *top, right, bottom, left.*

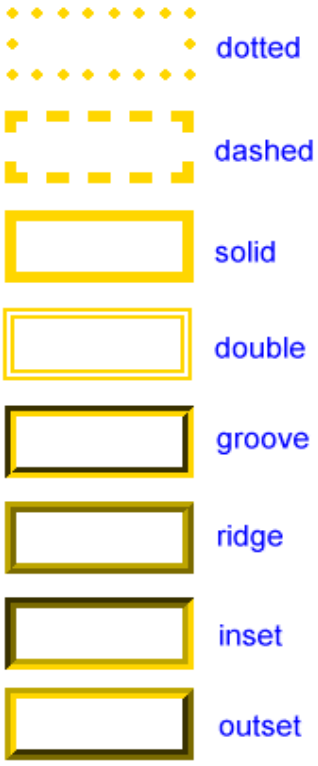
**`border`**

Параметр border може набувати значень:

`border-width` – thin, medium і thick, або числове значення в пікселях:


`border-color` – колір рамки, заданий будь-яким способом, наприклад, "#123456", "rgb (123,123,123)" або "yellow".

`border-style` (none або hidden – без рамок) може набувати таких значень:




Потрібно зауважити, що кожна сторона рамки може мати свій колір, стиль, ширину.

**`height`** – висота елемента,

**`width`** – ширина елемента, задані за допомогою певної одиниці виміру.

### 3.3 Одиниці виміру

**px** (піксель) – це базова, абсолютна одиниця виміру. Доступні ще декілька застарілих (похідні від *px*):

– 1mm (мм) = 3.8px;
– 1cm (см) = 38px;
– 1pt (типографський пункт) = 4/3 px;
– 1pc (типографська піка) = 16px.

Приваблює чіткість і зрозумілість цих одиниць виміру, але інші одиниці вимірювання – в деякому сенсі «могутніші», вони є відносними і дозволяють встановлювати співвідношення між різними розмірами.

**em** – відносно шрифтів (визначаються за поточним контекстом).

1 em – це поточний розмір шрифту. Можна брати будь-які пропорції відносно поточного шрифту: 2em, 0.5em і т. п.

**%** – проценти, відсотки від значення властивості «батька» з тією ж назвою (але не завжди), тобто, % від розміру шрифта «батька». Водночас можна вказати, відносно чого береться відсоток:

`margin-left` – відсоток береться від ширини батьківського блока, а не від його *margin-left;*

`line-height` – відсоток береться від поточного розміру шрифту, а не від *line-height* «батька»;

`width/height` – відсоток береться від ширини / висоти «батька», але за *position*: *fixed*, відсоток береться від ширини / висоти вікна (а не «батька» і не документа);

. . .

Приклади:

| Код | Вигляд на сторінці |
|---|---|
| `<div style="font-size: 24 px;>`<br>`  Студент ...`<br>`  <div style="font-size: 24 px;">`<br>`  Він і в Африці студент!`<br>`  </div>`<br>`</div>` | Студент ...<br>Він і в Африці студент! |
| `<div style="font-size: 24 px;>`<br>`  Студент ...`<br>`  <div style="font-size: 1.5em;">`<br>`  Він і в Африці студент!`<br>`  </div>`<br>`</div>` | Студент ...<br>Він і в Африці студент! |
| `<div style="font-size: 24 px;>`<br>`  Студент ...`<br>`  <div style="font-size: 150%;">`<br>`  Він і в Африці студент!`<br>`  </div>`<br>`</div>` | Студент ...<br>Він і в Африці студент! |

**rem** – задає розмір щодо розміру шрифту елемента <*html*>

Приклади:

| Код | Стиль і вигляд на сторінці |
|---|---|
| `<div style="font-size: 36 px; "{>`<br>`<ul><li> Ой, у лузі червона калина`<br>`<ul><li> похилилася`<br>`<ul><li>Чомусь наша славна Україна`<br>`<ul><li>зажурилася....`<br>`</li></ul>`<br>`</li></ul>`<br>`</li></ul>`<br>`</div>` | `li {font-size: 0.9em;}`<br>• Ой, у лузі червона калина<br>◦ похилилася<br>▪ Чомусь наша славна Україна<br>▪ зажурилася....<br>`li {font-size: 0.9rem; }`<br>• Ой, у лузі червона калина<br>◦ похилилася<br>▪ Чомусь наша славна Україна<br>▪ зажурилася.... |

**vw, vh, vmin, vmax** – розміри встановлюються відносно екрана:

vw – 1% ширини вікна;

vh – 1% висоти вікна;

vmin – найменше з (vw, vh) (в IE9 позначається vm);

vmax – найбільше з (vw, vh);

Ці значення були створені, насамперед, для підтримки мобільних пристроїв.

### 3.4 Керування контентом блока

#### 3.4.1 Властивість `overflow` – переповнення контенту

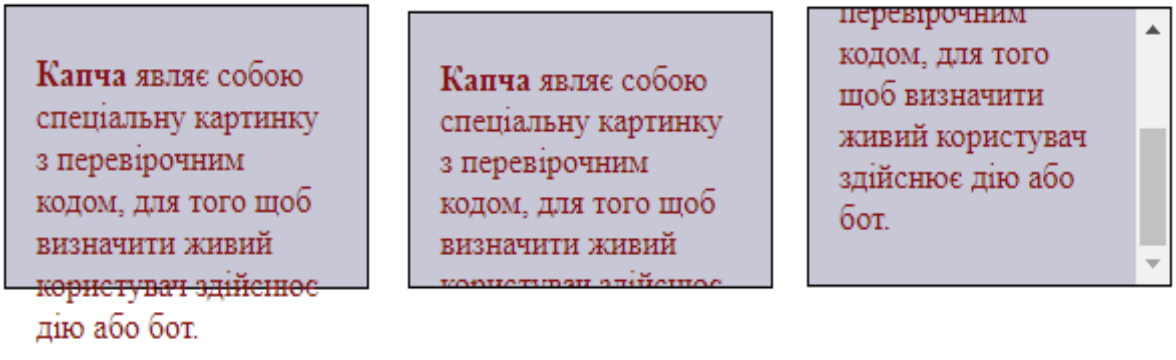
Ця властивість відповідає за поведінку контенту у випадку, якщо вміст контенту виходить за границі блока.

**`overflow:visible;`** – відображається весь вміст блока, навіть за межами встановленої висоти і ширини (встановлюється за замовчуванням);

**`overflow:hidden;`** – відображається тільки область всередині блока, решту буде приховано;

**`overflow:scroll;`** – завжди додаються смуги прокрутки, навіть якщо контент поміщається;

**`overflow:auto;`** – смуги прокрутки додаються тільки за необхідності.




`overflow: visible;` `overflow: hidden;` `overflow: auto;`

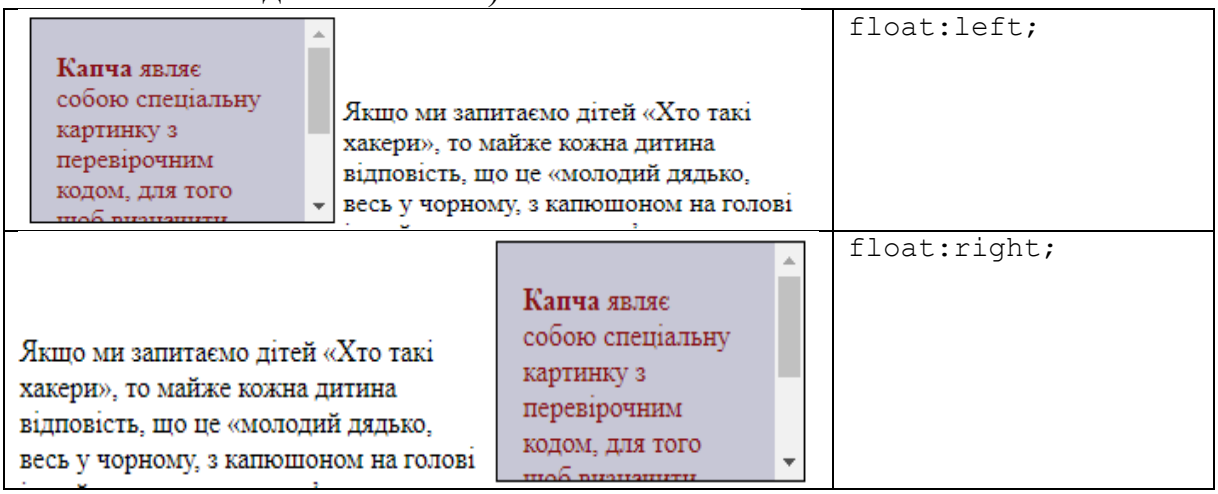
### 3.4.2 Властивість `float` – обтікання елементів

Ця властивість задає те, за яким краєм (лівим або правим) буде вирівняний ***сам елемент***. З протилежного боку його будуть обтікати інші елементи – текст і блоки.

**`float:left;`** – вирівнювання за лівою стороною, обтікання справа,

**`float:right;`** – вирівнювання за правою стороною, обтікання зліва,

**`float:none;`** – вирівнювання НЕ задається (потрібне, щоб скинути раніше задане значення).




### 3.4.3 Властивість `position` – позиціонування

Ця властивість вказує позицію відповідного елемента на сторінці. Наприклад, саме завдяки цій властивості закріплюють заголовок у певному місці. Існує декілька значень, які може набувати ця властивість.

**`position:static`** – за замовчуванням, в порядку опису;

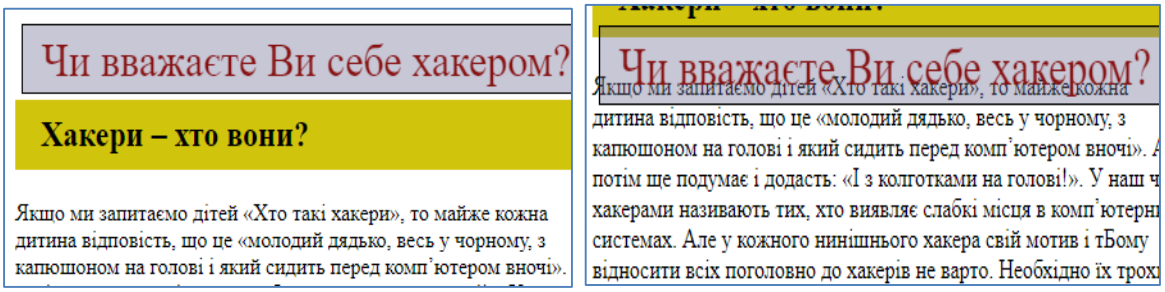
**`position:absolute`** – положення блока задається *відносно сторінки* або відносно елемента з *position:relative*, в який він вкладений, за допомогою *top, right, bottom, left*. Такі блоки ніби «ивалюються» з потоку і живуть своїм власним життям.

**`position:fixed`** – задається так само, як і абсолютна, але блок прикріплюється не до сторінки, а до екрана – *під час прокручування сторінки він залишається на місці*;

**`position:relative`**

- положення елемента встановлюється щодо його вихідного місця в потоці за допомогою *left, top, right і bottom*;
- працює як «батько», відносно якого можна задавати абсолютну позицію блокам, які в нього вкладені.

Наприклад, за використання **`position:absolute`** результат такий: зліва – блок стоїть на місці, справа – блок залишається під час прокручування сторінки:




### 3.5 Додаткові ефекти

#### 3.5.1 Псевдокласи

Це ключові слова, які додаються до селектора. Ми вже використовували псевдокласи до посилань. Так само їх можна застосовувати і до інших елементів на web-сторінках.

Стилі, задані за допомогою псевдокласів, застосовуються тільки для певних станів.

Найчастіше використовують такі псевдокласи стилів:

**:Hover** – з'являється під час наведення мишки;

**:Active** – з'являється під час натискання на елемент;

**:Focus** – з'являється під час фокусування на елементі (наприклад, коли вибрано поле введення тексту).

Приклад.

```
button {   border: 1px solid black;
padding: 8px 16px;
background: #c9c9c9;
}
button:hover { background: #e9e9e9; }
```

Перелік стандартних наперед заданих псевдокласів:

**:root:** дозволяє вибрати кореневий елемент веб-сторінки, напевно найменш корисний селектор, тому що на правильній веб-сторінці кореневим елементом практично завжди є елемент <html>;

**:link:** застосовується до посилань і являє собою посилання у звичайному стані, за яким ще не здійснено перехід;

**:visited:** застосовується до посилань і являє собою посилання, яким користувач вже переходив;

**:active:** застосовується до посилань і подає посилання в той момент, коли користувач здійснює за ним перехід;

**:hover:** являє собою елемент, на який користувач навів покажчик миші. Застосовується переважно до посилань, проте може також застосовуватись і до інших елементів, наприклад, до параграфів;

**:focus:** являє собою елемент, який отримує фокус, тобто, коли користувач натискає клавішу табуляції або натискає кнопкою миші на поле введення (наприклад, текстове поле);

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

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

**:empty:** вибирає елементи, які не мають вкладених елементів, тобто порожні.

Існує ще цілий ряд псевдокласів, призначених для спеціальних цілей: псевдокласи дочірніх елементів, псевдокласи форм, а також псевдоелементи.

### 3.5.2 Анімація (animation)

Універсальна властивість animation – загальна назва восьми властивостей анімації:

**animation-name –** встановлює одну або кілька анімацій, які застосовуються до елемента. Являє собою ім'я, пов'язане з правилом *@keyframes*, воно задає послідовність руху;

**animation-duration –** задає час в секундах або мілісекундах, скільки має тривати один цикл анімації. За замовчуванням значення дорівнює 0s, це означає, що ніякої анімації немає;

**animation-timing-function –** встановлює, відповідно до якої функції часу має відбуватися анімація кожного циклу між ключовими кадрами;

**animation-delay –** встановлює час очікування перед запуском циклу анімації;

**animation-iteration-count –** властивість визначає, скільки разів програвати цикл анімації до її зупинення;

**animation-direction –** встановлює напрямок руху анімації;

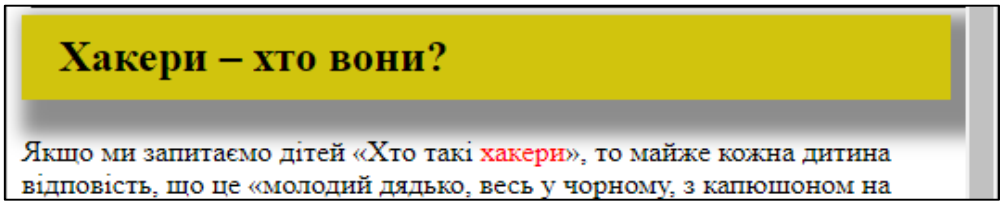
**animation-fill-mode –** визначає, які стилі мають застосовуватися до елемента, коли анімація не програється;

**animation-play-state –** властивість визначає, програвати анімацію або поставити її на паузу.

Наприклад, додамо в стиль для заголовка H2 рядок для анімації:

```
h2 {
    background: rgb(209, 196, 13);
    padding: 10px 20px 10px 20px;
    animation: move 1.15s ease-in 0s infinite
               alternate paused none;
    }
h2:hover { box-shadow: 10px rgba(0,0,0,0.45); }
@keyframes move {
        from {     top: 16px;
                   left: 16px;
        }
        to {       top: 250px;
                   left: 350px;
}     }
```

Як результат отримуємо бажаний ефект (поява тіні) під час наведення мишею над заголовком:




### 3.5.3 Переходи (`transition`)

CSS властивість *transition* встановлює ефект переходу між двома станами елемента. Використовується разом з псевдокласом *:hover* або :*active* або якщо стан елемента був динамічно змінений за допомогою JavaScript.

Властивість *transition* може отримувати 6 значень:

**`transition-property`** – ім'я властивості, за якою ми будемо слідкувати. І коли значення цієї властивості зміниться, запуститься ефект переходу;

**`transition-duration`** – тривалість ефекту переходу;

**`transition-timing-function`** `[ease/linear/ease-in/ease-out/ease-in-out/...]` – визначає криву (функцію) ефекту переходу;

**`transition-delay`** – затримка ефекту переходу;

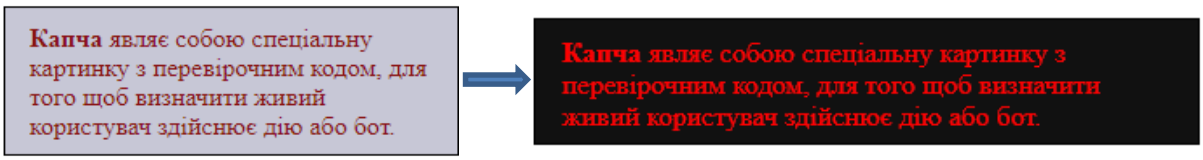
**`initial`** – встановлює властивість у значення без задання;

**`inherit`** – вказує на спадковість властивості від свого батьківського елемента (якщо відповідна властивість встановлена).

Приклад. Додамо у таблицю стилів для блока псевдокласу .p2 таке:

```
.p2 { width: 250px;
    height: 70px;
    border: 1px solid black;
    color: rgb(136, 19, 19);
    background-color: rgba(105, 105, 146, 0.377);
    margin: 5px 5px 5px 5px;
    padding: 10px 15px 10px 15px;
    transition: .65s ease-out .75s;
}
.p2:hover{  width: 350px;
    height: 50px;
    background-color: #111; color: red;
}
```

Тоді побачимо, що за наведення на вказаний блок він повільно збільшується і змінює колір фону і тексту:



### 3.5.4 Трансформація блоку (`transform`)

***Змінити розмір*** можна зробити двома способами.

Перший – це під час наведення на блок змінювати його ширину і висоту.

Другий варіант – використання **`transform: scale()`**. В дужках вказують два значення: розмір по горизонталі і по вертикалі. За замовчуванням значення дорівнюють одиниці, так що якщо записати 2, то розмір збільшитися в два рази. Також можна записувати з десятковими частками. Відповідно, за бажання можна і зменшувати елемент. Наприклад:

`transform: scale(1.5, 1.5)` – збільшить елемент в півтора рази;

`transform: scale(0.5, 0.5)` – зменшить в два рази;

`transform: scaleX(2)` – збільшить ширину блока в два рази.

***Поворот*** здійснюється за допомогою тих самих трансформацій, але вже за допомогою ***rotate***, значення задається у градусах. Наприклад,

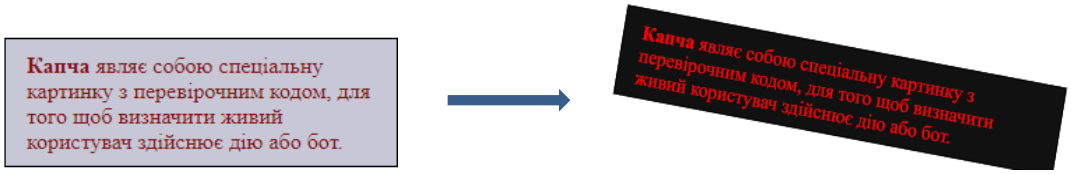
`transform: rotate(90deg)` – поверне елемент на 90° за годинниковою стрілкою;

`transform:rotate(-120deg)` – поверне на 120° проти годинникової стрілки, тобто, фактично, на 240 ° за годинниковою стрілкою.

Так, наприклад, додання такого рядка в стиль *.p2:hover* :

```
transform: rotate(10deg);
```

приведе до ефекту повороту усього блока на 10 градусів:




***Переміщення блоку.*** Для цього використовується параметр трансформації **`translate`**. До нього може додаватися X або Y для задання зсуву тільки з одного конкретного боку, якщо ж писати просто *translate*, то спочатку вказується зсув по горизонталі, потім – по вертикалі. Допускається використання від'ємних значень.

`transform: translateX(-100px)` – змістить блок на 100 пікселів вліво;

`transform: translate(150px, -200px)` – на 150 пікселів вправо і на 200 пікселів вгору.

***Нахил*** задається з допомогою параметру **`skew`**. Також можна задавати тільки з одного боку, використовуючи X і Y. Наприклад,

```
transform: skewX(15deg);
```

### 3.5.5 Реалізація висхідного меню

Дуже часто на web-сторінках виникає необхідність створити висхідне меню. Досить простий спосіб для досягнення цього – використати можливості CSS. Нехай, наприклад, у нас є елемент (кнопка), за наведення на який має з'явитися підменю.

```
<div class="menuMain">
    <button class="menu">Складові інформаційної
безпеки</button>
    <div class="menuPopup">
      <a href="#">Доступність</a>
      <a href="#">Цілісність </a>
      <a href="#">Конфіденційність</a>
    </div>
  </div>
```

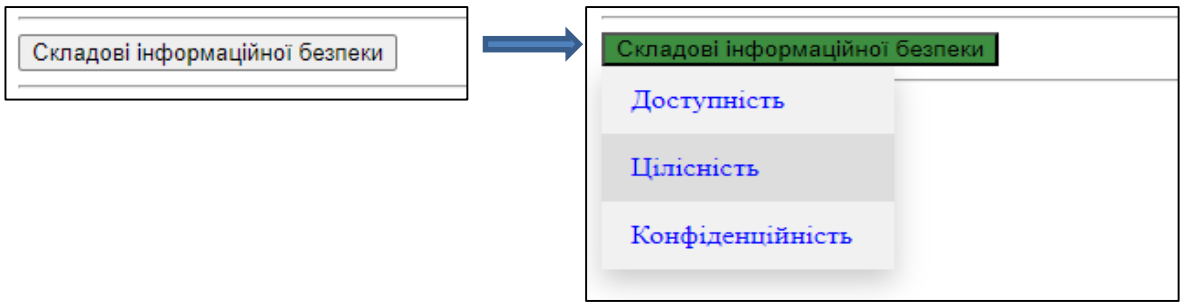
Стиль (клас *menuPopup*) для початку містить параметр *display: none*. Тобто, спочатку цей блок буде невидимим, а в активному стані він стає видимим *display: block*.

```
.menuPopup {   display: none;
    border-radius: 10;
    border-color: gold;
    position: absolute;
    background-color: #f1f1f1;
    min-width: 160px;
    box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
    z-index: 1;
}
.menuPopup a {display: block;
    color: blue;
    padding: 12px 16px;
    text-decoration: none;
}
```

І, нарешті, додаємо деякі ефекти за наведення миші над елементами:

```
.menuPopup a:hover {background-color: #ddd;}
.menuMain:hover .menuPopup {display: block;}
.menuMain:hover .menu {background-color: #3e8e41;}
```

Як результат маємо ефект висхідного меню:




### 3.5.6 Реалізація підвалу (футера), прикріпленого до низу

Для того, що створити футер і прикріпити його до низу сторінки, рекомендується:

- все тіло сторінки оформити у вигляді блока <*div*>;
- основну частину сторінки також оформити у вигляді блока <*div*>;
- за необхідності розбити основну частину на заголовок і контент;
- після основної частини створити блок підвалу <*footer*>;
- для кожної з частин підібрати свій стиль;
- використовуючи технологію *flexbox*, прикріпити *footer* до низу.

У коді це може виглядати таким чином:

```
<body>
    <div class="wrapper">
        <div class="content">
            <header class="header">
                <h2>ЗАГОЛОВОК</h2>
            </header>
            <section class="base">
                    К о н т е н т (основна частина)
            </section>
        </div>
        <footer class="footer">
           Підвал – футер
        </footer>
    </div>
</body>
```

Тепер залишається коректно описати стилі цих частин:

```
html,body { height: 100%; }
.wrapper{ display: flex;
          flex-direction: column;
          min-height: 100%;
        }
.content {  flex: 1 0 auto; }
.header  {  height: 100px; background-color:khaki;  }
.base {    height: 500px;
           background-color:rgb(182, 193, 230) ;
       }
.footer {   height: 100px;
            flex: 0 0 auto;
            background-color:rgb(82, 80, 62) ;
        }
```

### 3.5.7 Реалізація заголовка, прикріпленого до верху

Дуже часто виникає необхідність у верхній частині сторінки розташувати, наприклад, панель навігації, причому так, щоб вона не прокручувалась разом з основним контентом сторінки. Цього можна легко досягнути, оформивши «нерухому» частину в окремий блок і задати цьому блоку властивість: *position: fixed*. Якщо дотримуватись семантичної верстки сайту, то ця частина має бути в блоці *<header>…</header>*.

Так, у нашому попередньому прикладі, цей блок може виглядати так:

```
.header {
     margin-top: 80px;
     position: fixed;
     background-color:khaki;
    }
```

Як результат, бачимо, що основний текст сторінки прокручується, а заголовок залишається на місці.

За допомогою CSS можна реалізувати на web-сторінках ще багато цікавих ефектів: зміна прозорості, зміна кольору тощо (див. додаткові джерела).

### 3.6 Технологія *flexbox*

*Flexbox* (гнучкі коробки) – це відносно новий режим розмітки в CSS3, призначений для поліпшення вирівнювання, напрямку та порядку елементів в контейнері, навіть якщо він є динамічним або з невідомими розмірами. Це найважливіша особливість – можливість змінювати ширину або висоту дочірніх елементів в блоці, щоб найкращим чином заповнити обсяг контейнера, доступний за різних розмірів екрану.

*Flexbox* – це режим розмітки, створений для впорядкування елементів на сторінці таким чином, щоб вони вели себе передбачувано для випадків адаптивності сторінки під різні розміри екрану і для різних пристроїв.

Нині *flexbox* підтримується практично всіма сучасними браузерами, включно й Android та iOS.

Розберемось з цією технологією на прикладі.

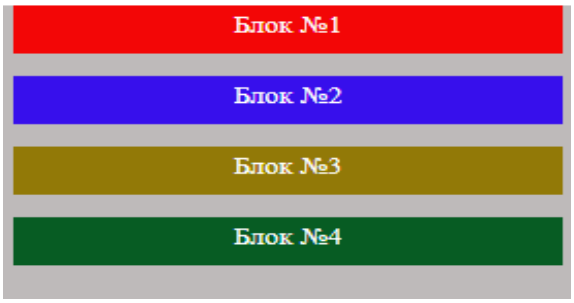
Нехай у нас є чотири блоки <*div*>:

```
<div class="bigblock">
   <div class="smallblock" id="B1"><p >Блок №1</p></div>
   <div class="smallblock" id="B2"><p >Блок №2</p></div>
   <div class="smallblock" id="B3"><p >Блок №3</p></div>
   <div class="smallblock" id="B4"><p >Блок №4</p></div>
 </div>
```

Для зрозумілості і красоти пофарбуємо їх в різні кольори і задамо ще деякі додаткові стилі:

```
bigblock {
  width: 50%;  height: 450px;
  margin: auto;
  background-color: #bebaba;
}
.smallblock {
  margin: 5px;
  height: 30px;
}
#B1{background-color: #f30606;}
#B2{background-color: #370fec;}
#B3{background-color: #927907;}
#B4{background-color: #075c23;}
p { padding: 5px;    color: #fff;
    font-size: 14px; text-align: center;
}
```

У кожного  є властивість *display: bloc*k. Тому кожен блок займає всю ширину рядка і блоки йдуть один за одним.

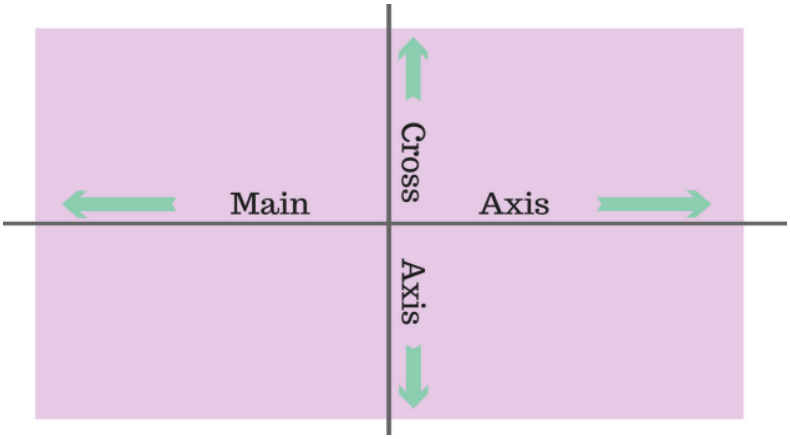


Для роботи з CSS Flexbox потрібно зробити контейнер *bigblock* flex-контейнером, тобто додати властивість: ***display: flex***.

Тепер у блоків (флексів) з'явилася властивість *flex-контекст*, яка надалі дозволить керувати ними набагато простіше, ніж з використанням стандартного CSS.

### 3.6.1 flex-direction – напрям розташування блоків

У головного контейнера *bigblock* є дві осі: головна та перпендикулярна їй.



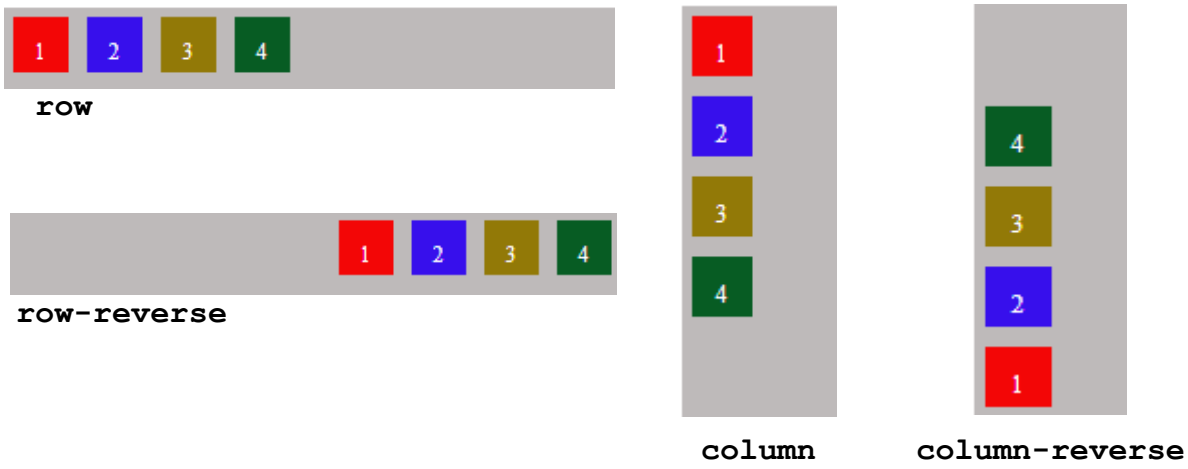

Надаючи різні значення властивості *flex-direction*, можемо керувати тим, вздовж якої осі і за яким напрямом розташовуються блоки:

**`row`** – головна вісь спрямована, як і орієнтація тексту, за замовчуванням зліва направо. Якщо значення *dir* задано як *rtl*, напрям осі йде справа наліво;

**`row-reverse`** – схоже на значення row, але міняються місцями початкова та кінцева точки та головна вісь спрямована праворуч наліво;

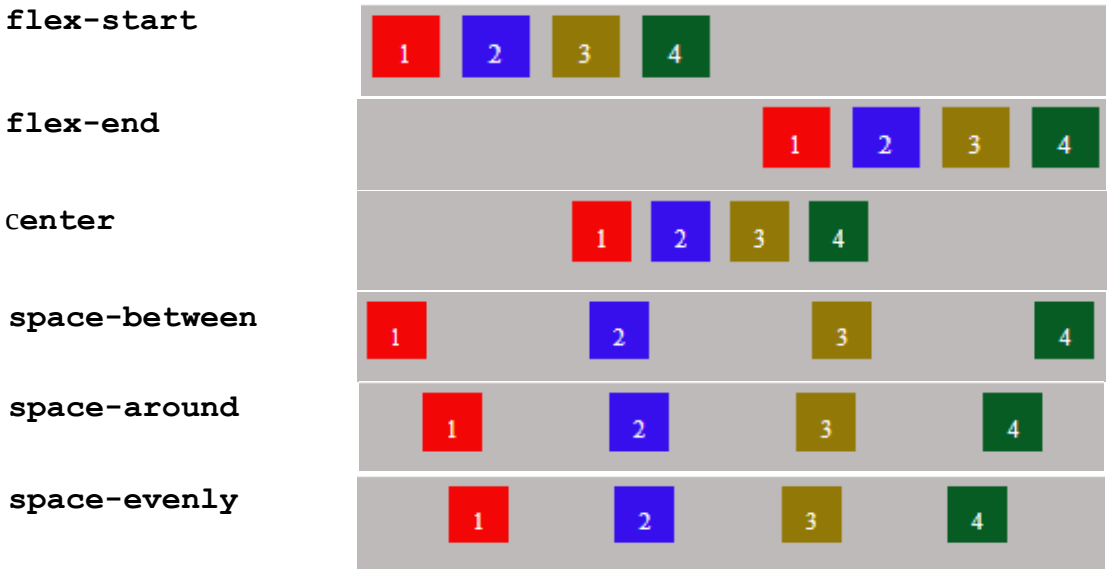
**`column`** – головна вісь розташовується вертикально та спрямована зверху вниз;

**`column-reverse`** – головна вісь розташовується вертикально, але змінюється положення початкової та кінцевої точок і вісь спрямована знизу вверх.



### 3.6.2 `justify-content` – вирівнювання по головній осі

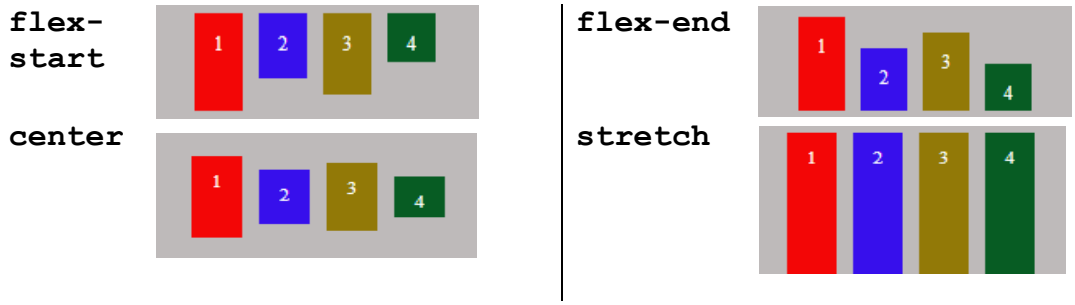
**`flex-start`** – флекси притиснуті до початку рядка;
**`flex-end`** – флекси притиснуті до кінця рядка;
**`center`** – флекси вирівнюються по центру рядка;
**`space-between`** – флекси рівномірно розподіляються по всьому рядку. Перший і останній елемент притискаються до країв контейнера;
**`space-around`** – флекси рівномірно розподіляються по всьому рядку. Порожній простір перед першим і після останнього елементів дорівнює половині простору між двома сусідніми елементами;
**`space-evenly`** – флекси розподіляються так, що відстань між будь-якими двома сусідніми елементами, а також перед першим і після останнього була однаковою.




### 3.6.3 `align-items` - вирівнювання

*Якщо justify-content* працює з головною віссю, то *align-items* працює з віссю, перпендикулярною до головної осі. Можливі такі значення:
**`flex-start`** – флекси вирівнюються на початку поперечної осі контейнера;
**`flex-end`** – флекси вирівнюються наприкінці поперечної осі контейнера;
**`center`** – флекси вирівнюються за лінією поперечної осі;
**`baseline`** – флекси вирівнюються за їхньою базовою лінією;
**`stretch`** – флекси розтягуються, щоб зайняти весь доступний простір.

Для демонстрації візьмем блоки різного розміру.



### 3.6.4 `align-self` - вирівнювати окремого елемента

Значення цієї властивості такі самі, як і у властивості *align-items*, але вирівнювати можна будь-який окремий елемент контейнера.

Наприклад, усі елементи вирівняні по верхньому краю, а четвертий блок – по центру.

```
#B4{
    background-color: #075c23;
    align-self: center;
    height:  30px;
    }
```


### 3.6.5 `align-content` – вирівнювання за наявності вільного простору

**`flex-start`** – рядки розташовуються на початку поперечної осі. Кожен наступний рядок йде нарівні з попереднім;

**`flex-end`** – рядки розташовуються, починаючи з кінця поперечної осі. Кожен попередній рядок йде врівень з наступним;

**`center`** – рядки розташовуються по центру контейнера;

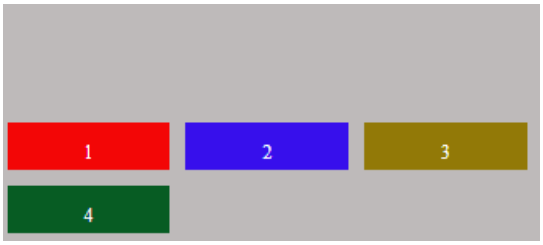
**`space-between`** – рядки рівномірно розподіляються в контейнері і відстань між ними однакова;

**`space-around`** – рядки рівномірно розподіляються таким чином, щоб простір між двома сусідніми рядками був однаковим. Порожній простір перед першим рядком і після останнього рядка дорівнює половині простору між двома сусідніми рядками;

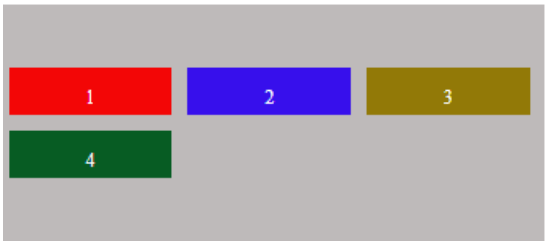
**`space-evenly`** – рядки розподіляються рівномірно. Порожній простір перед першим рядком та після останнього рядка має ту саму ширину, що й у інших рядків;

**`stretch`** – рядки рівномірно розтягуються, заповнюючи вільний простір.

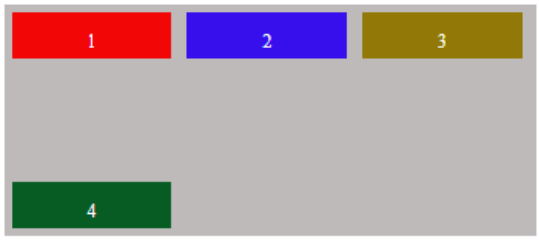

**`flex-start`**



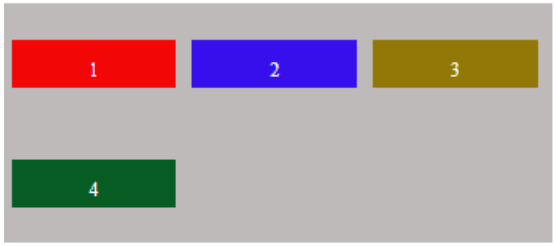

**`flex-end`**



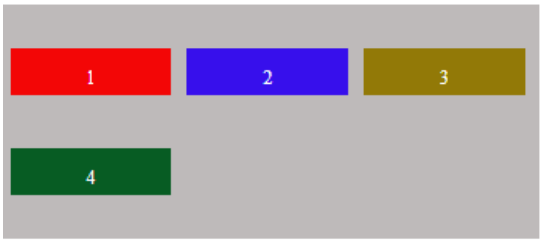

**`center`**




**`space-between`**




**space-around**




**space-evenly**

### 3.6.6 Інші властивості

Існує ще цілий ряд властивостей, з якими можна познайомитись за допомогою вивчення додаткових інформаційних джерел.

**flex-wrap** – вказує, потрібно флексам розташовуватися в один рядок чи можна зайняти кілька рядків. Якщо перенесення рядків допускається, то властивість також дозволяє контролювати напрямок, у якому викладаються рядки. Значення за замовчуванням: *nowrap*.

`nowrap` – флекси вишиковуються в одну лінію;

`wrap` – флекси вишиковуються в кілька рядків, їх напрямок задається властивістю flex-direction;

`wrap-reverse` – флекси вишиковуються в декілька рядків, у протилежному напрямку *flex-direction*.

**flex-flow** є скороченою властивістю для окремих властивостей *flex-direction* та *flex-wrap*. Значення за замовчуванням: *row nowrap*.

**order** – визначає порядок виведення флексів усередині флекс-контейнера. Елементи розташовуються згідно зі значеннями властивості *order* від меншого до більшого. За однакових значень *order* елементи виводяться у порядку, як вони з'являються у вихідному коді. Значення за замовчуванням: 0.

**flex-grow** – визначає коефіцієнт зростання *flex* для заданого числа. Негативне значення не валідне. Значення за замовчуванням: 0.

**flex-shrink** – задає коефіцієнт ущільнення *flex* із заданим числом. Негативне значення не валідне. Значення за замовчуванням: *1*.

**flex-basis** – визначає основу флексу, що є початковим розміром елемента. Схоже на властивості *width* та *height*, до яких додається вміст елемента. Значення за замовчуванням: *auto*.

**flex** – замінює разом *flex-grow, flex-shrink* та *flex-basis*. Значення за замовчуванням: *0* (*grow*) *1 (shrink) auto (basis).*

З технологією *flexbox* у повному обсязі пропонується ознайомитись за додатковими інформаційними джерелами, яких досить багато у мережі інтернет.

### Контрольні запитання

1. Що означає боксова модель CSS?
2. Назвіть основні області боксової моделі і їх призначення.

3. Як змінити параметри блоків (задати ширину, висоту, відступи тощо).
4. Як здійснюється керування переповненням блоків, обтіканням блоків та їх позиціонуванням?
5. Як можна реалізувати анімацію елементів сторінки?
6. Які одиниці вимірювання використовують в CSS?
7. Яким чином можна задавати кольори?
8. Які стандартні псевдокласи Ви можете назвати? Для чого їх використовують?
9. Назвіть, які додаткові ефекти можуть бути застосовані на сторінках.
10. Як можна притиснути заголовок до верхнього краю сторінки?
11. Як можна закріпити підвал до нижнього краю сторінки?
12. В чому полягає суть технологій *flexbox*?
13. Наведіть основні властивості боксів і приклади їх використання.
14. Що вкладають у поняття адаптивності сайту?
15. Що таке медіазапити і брейкпоінти?

## Практична робота № 3

### *Мета роботи*

– Ознайомитись з поняттям і основними можливостями боксової моделі.
– Опанувати основні можливості керування блоками.

### *Порядок виконання роботи*

*Завдання 1. Застосування боксової моделі*

1. Внести в html-код одної або декількох сторінок доповнення для того, щоб використати блокову модель [1–3].
2. Застосувати для різних блоків різні параметри переповнення контексту блоку, обтікання та позиціонування, вирівнювання та розташування.

*Завдання 2. Використання додаткових візуальних ефектів*

1. Використати псевдокласи для підкреслення візуальних ефектів елементів сторінки у різних її станах [4].
2. Використати елементи анімації або інші цікаві візуальні ефекти: плавність, зміна розмірів, повороти, переміщення, нахили тощо [5–7].

*Завдання 3. Опанування технології flexbox*

1. Передбачити в панелі навігації (в меню) на основній сторінці перехід на окрему web-сторінку для виконання лабораторної роботи № 3 (тобто, створити для неї окрему html-сторінку).
2. Створити на цій сторінці не менше шести блоків, заповнених певною інформацією у графічному або текстовому вигляді (бажано за темою web-сайту). Ознайомитись з суттю технології *flexbox* [8–9].
3. Опанувати на практиці систему флексбокс, для чого спробувати використати почергово усі властивості блоків-флексів [10–11].

4. Спробувати розташувати блоки з основною інформацією на власній сторінці по-різному, змінюючи властивості в CSS.

** Завдання 4. Застосування способів надання web-сторінкам адаптивності*

1. Самостійно ознайомитись з поняттям адаптивної сітки для сайтів.
2. Дослідити можливості керування розміщенням блоків, використовуючи медіа-запити.
3. Використовуючи адаптивні сітки, змінити спосіб розташування блоків на сторінці, застосовуючи медіа-запити та брейкпоінти [12–13].

**Додаткові джерела**


1. Блочна модель CSS. [Електронний ресурс]: URL: https://www.w3schools.com/css/css_boxmodel.asp#.
2. Блочна модель CSS. [Електронний ресурс]: URL: https://webportal.com.ua/box-model-css/.
3. Псевдокласи CSS. [Електронний ресурс]: URL: https://html-css.co.ua/self-css/psevdoklasi/.
4. CSS: псевдоелементи і псевдокласи. [Електронний ресурс]: URL: https://hi-news.pp.ua/kompyuteri/6296-css-psevdoelementi-psevdoklasi.html.
5. Красиві css3 ефекти, які легко реалізувати. [Електронний ресурс]: URL: http://yoip.com.ua/krasivi-css3-efekti-yaki-legko-realizuvati/.
6. CSS-анімації. [Електронний ресурс]: URL: https://uk.javascript.info/css-animations.
7. Застосування анімацій CSS. [Електронний ресурс]: URL: https://webdoky.org/uk/docs/Web/CSS/CSS_Animations/Using_CSS_animations/#nalashtuvannia-animatsii.
8. Використання Flexbox в CSS3 для адаптивного дизайну. [Електронний ресурс]: URL: https://sebweo.com/vikoristannya-flexbox-v-css3-dlya-adaptivnogo-dizajnu/.
9. Як працює CSS Flexbox. [Електронний ресурс]: URL: https://petrov.net.ua/how-css-flexbox-works/.
10. Шпаргалка по Flexbox CSS. [Електронний ресурс]: URL: https://tpverstak.ru/flex-cheatsheet/.
11. A Complete Guide to Flexbox. [Електронний ресурс]: URL: https://css-tricks.com/snippets/css/a-guide-to-flexbox/.
12. Знайомство з CSS Grid Layout. [Електронний ресурс]: URL: https://css.in.ua/article/znayomstvo-z-css-grid-layout_12
13. Grid в CSS: повний посібник та довідник з верстки. [Електронний ресурс]: URL: https://highload.today/uk/grid-v-css-povnij-posibnik-ta-dovidnik-z-verstki/.

# 4 СУЧАСНІ ПРОГРАМНІ ЗАСОБИ ДЛЯ РОЗРОБЛЕННЯ WEB-СТОРІНОК

## 4.1 Сучасні інструменти для верстування сайту

Створити сайт можна по-різному:.

- по-перше, можна використати ***конструктор*** сайтів;
- по-друге, отримати завдання на розробку ***від дизайнера***, а потім розробляти сайт відповідно до отриманого завдання, не вникаючи у деталі;
- по-третє, можна ***самостійно*** розробити і шаблон сайту, і його наповнення.

### 4.1.1 Шаблон сайту

Шаблони – це спеціально розроблені зображення, підготовлені для того, щоб стати основою для сайтів різного призначення. Такі зображення зазвичай складаються з шарів, що дозволяє змінювати (видаляти) більшість наявних на них елементів. У формі окремих файлів їх зазвичай можна зустріти з розширенням *.psd*, що дозволяє редагувати їх в графічних редакторах типу *Photoshop* і аналогів.

Отже, *psd-шаблон* – вихідний матеріал для верстки сайту повністю відображає не тільки дизайн майбутнього ресурсу, але і його розміри, схему розташування всіх елементів

Сьогодні в мережі існує достатня кількість сервісів – графічних редакторів, що дозволяють працювати самостійно з шаблонними. Наприклад, сервіс *Canva* пропонує тисячі шаблонів, або *Figma* – крос-платформний онлайн-сервіс для дизайнерів інтерфейсів і веб-розробників (для розробки інтерфейсів в онлайн-додатку) (рис. 4.1).

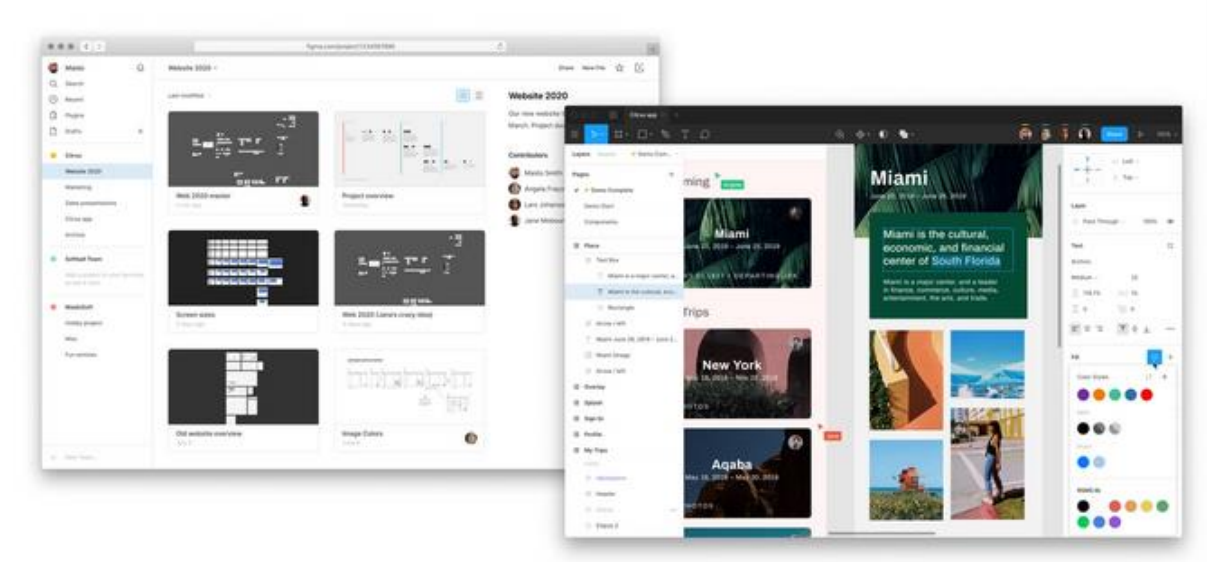



Рисунок 4.1 – Фрагменти інтерфейсу програми Figma

### 4.1.2 Конструктори сайтів

Конструктор сайтів – це сервіс, використовуючи який можна швидко створювати інтернет-ресурси навіть без знання основ програмування. Як правило, це система з набору інструментів, яка дозволяє будувати сайт (візитка, крамниця, …) і адмініструвати їх без будь-яких спеціалізованих знань. Але все не так просто. Використання конструкторів має свої переваги і недоліки. До *переваг* відносять таке:

- щодо витраченого часу, то завдяки конструктору сайт можна створити всього за декілька годин, вибравши шаблон і, за необхідності, відредагувати його під свої потреби;
- з погляду фінансів конструктор веб-сайтів дозволяє впоратися з цим завданням без залучення фахівців, що економить гроші для створення сайту;
- не потрібно знати мови програмування та тонкощі розмітки. Потрібно лише заповнити деякі форми та завантажити картинки, що в деяких випадках навіть простіше зробити, ніж в соціальних мережах.

*Недоліками* використання конструкторів є те, що:

- як результат буде отримано шаблонний дизайн, який можна лише частково змінити, тому в інтернеті буде багато подібних сайтів;
- найчастіше конструктор сайтів генерує громіздкий HTML-код, який містить багато зайвих рядків. Внаслідок цього сторінка, яка могла б важити зовсім мало, має великий обсяг і довго завантажується;
- не завжди можна відредагувати код вручну, оскільки він перенасичений тегами, стилями та іншими елементами. Програмістові простіше зробити новий сайт, ніж розбиратися в рядках заплутаного коду;
- структура та наповнення сайту обмежені його шаблоном. Звичайно, в межах запропонованих рішень;
- створений проєкт буде прив'язаний до конкретного майданчика. Обмежені можливості щодо розширення та розвитку проєкту. Можна підключити лише ті модулі, що доступні на майданчику. А значить перенести проєкт на інший майданчик досить не просто.

***Безкоштовні конструктори сайтів***

*Weblium* – надзвичайно простий у використанні конструктор сайтів, з унікальними стильними дизайнами, блоками та функціями. Безкоштовний, платні тарифні плани – від $5/місяць.

*Wix* – функціональна та гнучка платформа, для великої кількості кейсів пропонує широкий вибір шаблонів. Безкоштовний, платні тарифні плани – від $4.50/місяць.

*Хорошоп* – український конструктор сайтів для великих інтернет-магазинів. Тарифні плани – від 160 гривень на місяць.

*Jimdo* – конструктор без кастомізації дизайну та функціонального eCommerce. Особисті, корпоративні сайти та лендінги. Безкоштовний, тарифні плани від 110 гривень на місяць.

*Site123* – конструктор для лендингів, візиток та невеликих онлайн-магазинів. Безкоштовний, платні тарифні плани – від $9.8/місяць.

*Webnode* – доволі збалансований конструктор, тут можна створити корпоративний сайт, лендинг чи сайт-візитку. Безкоштовний, платні тарифні плани – від $3.95/місяць.

*Webflow* – професійний інструмент для створення унікальних сайтів без коду, складний в опануванні. Безкоштовна версія, платні тарифні плани – від $14/місяць.

### 4.1.3 CSS-фреймворки

Фреймворк (англ. *framework* – каркас) – це програмне середовище, яке спрощує та прискорює створення програмного забезпечення. За використання фреймворків пишеться лише код, який реалізує логіку, специфічну для конкретного продукту. Водночас не доводиться самостійно забезпечувати роботу з базою даних, автентифікацію, підтримку сеансів тощо. Тобто, CSS-фреймворк – це CSS-код, який написали одні розробники і виклали його для інших – економити час під час розробки, не вигадувати свій велосипед без потреби. Зазвичай фреймворки містять в собі:

– великий набір компонентів;
– можливість використання адаптивної сітки.

Одним з найпопулярніших CSS-фреймворків є *Bootstrap* (від розробників Twitter).

Звичайно, використання фреймворків має багато *позитивних рис*:

– набір готових UI-компонентів (швидкість розробки вище, більш красивий інтерфейс, якщо у розробників поганий смак чи час на розробку обмежений);
– більшість розробників з ними знайомі (простіше підтримувати код, новим членам команди простіше почати роботу);
– спільнота розробників (багато прикладів використання і навчальних матеріалів, невелика кількість багів, легко нагуглити рішення майже будь-якої проблеми).

Разом з тим, *негативні* риси у фреймворків також існують:

– зобов'язання працювати в рамках, заданих розробниками фреймворка (якщо виникає проблема, вирішення якої не передбачено в фреймворку, доводиться викручуватися і робити «фінти вухами», чим далі розвивається проект, тим складніше поміняти або позбутися фреймворка);
– зазвичай використовується тільки частина можливостей фреймворка;
– невикористаний код збільшує розмір сторінок, вони завантажуються повільніше;
– необхідно витрачати час на вивчення;
– шаблонність: сайти, створені на основі фреймворку, зазвичай не відрізняються індивідуальністю.

### 4.2 Поняття деплою, бекенду, фронтенду, хостингу

***Деплой*** – ***(deploy***) це розгортання – розміщення виконуваного коду на сервер, де він буде працювати. Це процедура запуску вебсайту або програми на сервері, хостингу. Деплой дозволяє кінцевим користувачам отримати доступ до ресурсів через мережу.

Причому цією мережею може бути як інтернет, так і будь-яка локальна мережа. Все залежить від вебресурсу.

***Фронтенд*** відповідає за той код, який виконується на стороні користувача, тобто на його пристрої – комп'ютері або телефоні:

- розмітка (HTML);
- верстка (CSS);
- програмування (JavaScript, PHP, …).

***Бекенд*** відповідає за серверну частину – обробку запитів користувача і зберігання даних:

- програмування: PHP, Python, Java і C# (використовуються в великих корпоративних проектах, де вони взаємодіють з громіздкими екосистемами), JavaScript;
- налаштування й адміністрування серверів;
- проектування баз даних та інше.

***Хостинг*** – сервіс, який розміщує код бекенду на сервері

Зазвичай хостинг – це платна послуга. В неї входить оренда місця на сервері та його потужностей.

Компанії, які надають хостинг, називаються *хостерами* або *хостинг-провайдерами*. Завдання цих компаній – зробити так, щоб сайт був цілодобово доступний користувачам.

*Хостинг потрібен для супроводження,* – зберігати файли із сайтом у цілодобовому доступі, керувати цими файлами та змінювати їх властивості. Для цього хостинг-провайдери встановлюють спеціальне обладнання – панелі керування.

Існує достатня кількість безкоштовних хостингів. Але пропозиція безкоштовного хостингу звучить занадто добре, щоб бути правдою. Надання послуг з веб-хостингу передбачає значні витрати для провайдера, і можна бути впевненим, що компанії, які пропонують безкоштовні хостинги, не роблять це просто так. Провайдер безкоштовного хостингу часто збирається використовувати сайт для створення прибутку для себе або для примусового відображення реклами. Або буде нав’язувати численні рекламні додатки (add-on). Якість послуг, які надають провайдери безкоштовного хостингу, також викликає занепокоєння.

Саме тому рекомендується обирати «майже безкоштовні» варіанти, замість безкоштовних – платні хостинги, ціни на які настільки низькі, що задовільнять навіть найбільш обмежені бюджети, наприклад, бюджетні тарифні плани від Hostinger. Як результат, якість хостингу буде набагато кращою.

### Контрольні запитання

1. Які основні поняття використовують під час розробки сайтів?
2. Для чого використовують конструктори сайтів?
3. Які позитивні і негативні наслідки використання конструкторів сайтів?
4. Назвіть відомі онлайн-сервіси для дизайну інтерфейсу сайтів.
5. Що таке CSS-фреймворки? Які функції вони виконують?
6. Що таке деплой сайтів?
7. Які завдання покладається на бекенд та фронтенд розробників сайтів?
8. Що таке хостинг? Чим займаються хостингові компанії?

### Практична робота № 4

***Мета роботи***

Ознайомитись з сучасними можливостями створення web-сторінок: конструкторами сайтів, фреймворками, програмами-дизайнерами сайтів.

***Завдання до виконання***

Використовуючи будь-який безкоштовний конструктор сайтів (бажано українського виробництва), підготувати і підключити до основної сторінки (можна з підвалу):

- власну особисту сторінку;
- власну візитку студента ВНТУ.

### Додаткові джерела


1. 21 кращий конструктор сайтів: повний огляд. [Електронний ресурс]: URL: https://hostiq.ua/blog/ukr/site-builders/.
2. Топ-6 кращих конструкторів сайтів у 2022 році. [Електронний ресурс]: URL: https://cityhost.ua/uk/blog/top-6-luchshih-konstruktorov-saytov-v-2022-godu.html.
3. Безкоштовні адаптивні шаблони веб-сайтів. [Електронний ресурс]: URL: https://www.templatemonster.com/ua/free-templates/website-template.html.
4. Галерея дизайнів. [Електронний ресурс]: URL: https://horoshop.ua/ua/design/.
5. FIGMA. [Електронний ресурс]: URL: https://www.figma.com/design/.
6. Bootstrap. [Електронний ресурс]: URL: https://getbootstrap.com/.
7. 8 Кращих БЕЗКОШТОВНИХ Хостингів у 2023 році. [Електронний ресурс]: URL: https://www.websiteplanet.com/uk/blog/.

# 5 БАЗОВІ КОНСТРУКЦІЇ МОВИ JAVASCRIPT. ОБ'ЄКТИ ЯДРА МОВИ JAVASCRIPT

## 5.1 Загальні відомості

### 5.1.1 Коротка історична довідка

Мову JavaScript було створено для того, щоб «оживити web-сторінки».

Програми цією мовою називаються *скриптами*. Їх можна писати прямо на сторінці в коді HTML і вони автоматично виконуватимуться в процесі завантаження сторінки. Скрипти надаються та виконуються як простий текст. Для запуску їм не потрібна спеціальна підготовка чи компілятор.

На початку створення мови JavaScript вона мала назву LiveScript. Але, оскільки на той час була дуже популярною мова програмування Java, було вирішено, що позиціонування нової мови як «молодшої сестри» Java допоможе в її популяризації. Згодом JavaScript стала повністю незалежною мовою програмування зі своєю специфікацією ECMAScript і зараз має мало спільного з Java. Її *особливості* у тому, що:

- застосовують для створення на web-сторінках інтерактивних елементів (для побудови меню, зміни зображень і т. д.);
- виконується на стороні користувача за допомогою браузера;
- підтримується всіма основними браузерами Web (Internet Explorer, Firefox, Netscape, Safari, Opera, Camino і т. д.);
- не потребує особливих програмних середовищ .

*На відміну від інших мов* програмування JavaScript:

- має можливість роботи з об'єктами, включно й визначення типу і структури об'єкта під час виконання програми;
- має можливість передавати і повертати функції як параметри, а також призначати їх змінній;
- має наявний механізм автоматичного узгодження типів;
- використовує автоматичний збір сміття;
- передбачає використання анонімних функцій.

Сьогодні JavaScript може виконуватися не тільки в браузері, але й на сервері або на будь-якому пристрої, який має спеціальну програму – рушій JavaScript.

### 5.1.2 Додання коду JavaScript на сторінку HTML

Три способи вбудовування коду JavaScript в код HTML.

1. Усередині тегів *<script> та </script>* безпосередньо у файлі **.html*:

```
<script type="text/javascript">
     . . .
     /*  тут код скрипта */
     . . .
</script>
```

2. В окремому файлі і зв'язатися з ним за допомогою тегів *<script> та </script>* :

```
<script type="text/javascript"
        src="scripts/JavaScriptFile.js">
</script>
```

3. Вбудувати код скрипта «на льоту», безпосередньо у тегах HTML. Цей спосіб використовується, як правило, під час обробки повідомлень. Наприклад,

```
<fieldset style="background-color: burlywood; color: blueviolet;">
  <legend style="text-align: center; font-family: Arial, sans-serif;">
     <b>ІНФОРМАЦІЯ ПРО УЧАСНИКА</b>
  </legend>
</fieldset>
```

### 5.2 Базовий синтаксис JavaScript

Код складається з інструкцій. Інструкції – це синтаксичні конструкції та команди, які виконують дії.

Після інструкцій ***крапку з комою*** можна ставити, а можна і пропустити. Часто крапку з комою пропускають, якщо є перенесення на новий рядок, але здебільшого її в кінці інструкцій ставлять (звичка від програмування іншими мовами, та й код виходить більш читабельним).

#### 5.2.1 Коментарі

***Однорядкові*** коментарі починаються з подвійної скісної //. Частина рядка після // вважається коментарем. Такий коментар може займати весь рядок або міститися після інструкції.

***Багаторядкові*** коментарі починаються зі скісної з зірочкою /* і закінчується зірочкою зі скісною */.

Вміст коментаря ігнорується, код ігнорується.

#### 5.2.2 Змінні

Щоб створити змінну, використовують ключові слова ***let*** і ***var***. *Var* є дещо застарілим, але насправді часто використовується. Наприклад,

```
var students = 19;
let quest = “Запитання”;
var message = 'Привіт!';
let user = 'admin', psw = 'p@ssw0rd';
```

Деяка різниця між ними все ж є:

а) *область видимості* змінних:
- **var** видима всередині всієї функції, в якій відбулося оголошення;
- **let** видима тільки всередині блока {...}, в якому вона була оголошена.

*б) Час видимості:*

– **var** – може бути доступна в коді аж до точки, в якій вона оголошена (в рамках загальних правил видимості змінних);
– **let** – видима тільки після оголошення;

*в) Змінна як лічильник циклу:*

– **var** діє протягом усього циклу і доступна навіть після його завершення;
– **let** – кожна ітерація циклу буде мати свою власну незалежну змінну.

***Локальні змінні*** – це змінні, оголошені в функції JavaScript і доступні тільки всередині функції, в якій вони оголошені. Під час виходу з цієї функції змінні знищуються.

***Глобальні змінні*** оголошуються поза функцією, і всі функції та скрипти можуть отримати до них доступ. Ці змінні знищуються в процесі закриття сторінки. Якщо оголосити змінну без використання *var* або *let*, то вона автоматично оголошується глобальною

### 5.2.3 Ідентифікатори змінних

Ідентифікатори – це назви змінних; до них застосовуються такі правила:

– імена змінних чутливі до регістра (*y* і *Y* – дві різні змінні);
– імена змінних мають починатися з літери або з одного із символів: «$» і «_»;
– ім'я змінної може складатися з будь-яких цифр і букв латинського алфавіту, а також символів «$» і «_»;
– не можна використовувати зарезервовані або ключові слова як імена змінних.

## 5.3 Типи даних

Мова JavaScript – нетипізована мова або «динамічно типізована». Йдеться про те, що типи даних визначені, але змінні не прив’язані до жодного типу. Наприклад,

```
let answer = ”admin”;
answer = 123;
answer = 12.345;
```

У JavaScript є вісім основних типів даних. Розглянемо деякі з них.

***Numbers***

Цей тип даних подає і цілі числа, і числа з рухомою точкою. Наприклад,

```
let n = 123;
n = 12.345;
```

До них можуть застосовуватись усі відомі арифметичні операції (+, /, *, +=, -=, /= , ++, --, =, %, ** – піднесення до степеня).

Окрім звичайних чисел, є так звані «спеціальні числові значення», що також мають відношення до цього типу даних:

**`Infinity (-Infinity)`** – являє собою математичну нескінченність ∞ (-∞). Це спеціальне значення, що є більшим за будь-яке число;

**`NaN`** – (Not a Number) являє собою помилку обчислення. Це є результат неправильної або невизначеної математичної операції.

### *BigInt*

Тип Number не може містити числа більші за $2^{53}$-1 (це число 9007199254740991), або менші за -$2^{53}$+1 для від'ємних чисел. Це технічне обмеження, спричинене їх внутрішньою реалізацією.

Для більшості потреб цього достатньо, але бувають випадки, коли потрібні дійсно великі числа, наприклад, для криптографії або мікросекундних часових міток (timestamps).

Значення з типом BigInt створюється через додавання *n* у кінець цілого числа, наприклад,

```
const bigInt = 1234567890123456789012345678901234567890n.
```

### *String*

Рядок у JavaScript має бути взятий у лапки.

```
let str = "Привіт";
let str2 = 'Одинарні лапки також дозволяються';
let phrase = `так можна вставляти ${str}`;
```

У JavaScript є три типи лапок:

- подвійні лапки: `"Привіт"`.
- одинарні лапки: `'Привіт'`.
- обернені лапки: `` `Привіт` ``.

Подвійні та одинарні лапки є «звичайними». Тобто немає ніякої різниці, які саме використовувати.

Обернені лапки є розширенням функціональності. Вони дають змогу вбудовувати змінні та вирази в рядок, беручи їх в `${…}`, наприклад:

```
let name = "Користувач";
alert(`Привіт ${name}у!`); // Результат: Привіт Користувачу!
```

або `alert(`результат: ${1 + 2}`); // Результат: 3`

Увага! У JavaScript немає символьного типу *char* (character). Є єдиний тип *string*. Рядок може містити нуль символів (бути пустим), один символ або більше.

### *Logical*

Логічний тип може набувати лише двох значень: *true* (істина) та *false* (хиба). Наприклад:

```
let nameFieldChecked = true;
let ageFieldChecked = false;
```

або як результат порівняння:

```
let isGreater = 4 > 1;
alert(isGreater); // Результат: true
```

Логічний тип може з’явитися за виконання ***операцій порівняння***:

| *Оп-р* | *Опис* | *Приклад* | *Результат (x=7)* |
|---|---|---|---|
| **==** | Перевіряє змінні чи значення на рівність. | x==7 | true |
| **===** | Перевіряє змінні або значення на рівність, враховуючи тип змінної | x===7<br>x==="7" | true<br>false |
| **!=** | Перевіряє, чи відрізняються змінні або значення | x!=9 | true |
| **>** | Порівняння двох змінних або значень | x>13 | false |
| **<** | | x<13 | true |
| **>=** | | x>=13<br>x>=7 | false<br>true |
| <= | | x<=13<br>x<=7 | true<br>true |

Логічні вирази можуть бути об’єднані ***логічними операціями***:

| *Оператор* | *Значення* | *Приклад* | *Результат* |
|---|---|---|---|
| **&&** | І | (x==2 && y==9)<br>(x==3 && y==9) | true<br>false |
| **\|\|** | АБО | (x==2 \|\| y==8)<br>(x==3 \|\| y==9)<br>(x==5 \|\| y==6) | true<br>true<br>false |
| **!** | НЕ | !(x==3) | true |

### *Null*

Спеціальне значення *null* не належить до жодного з описаних вище типів. Воно формує окремий власний тип, який містить лише значення *null*:

```
let psw = null;
```

В JavaScript *null* не є «посиланням на неіснуючий об’єкт» або «покажчиком на null», як може бути в інших мовах програмування. Це лише спеціальне значення, яке подає «нічого», «порожнє» або «невідоме значення».

### *Undefined*

Це спеціальне значення, яке представляє власний тип, подібний до *null*, і означає, що «значення не присвоєно».

Якщо змінна оголошена, але їй не присвоєно якесь значення, тоді значення такої змінної буде *undefined*:

```
let psw;
alert(psw); // Результат: undefined
```

***Object***

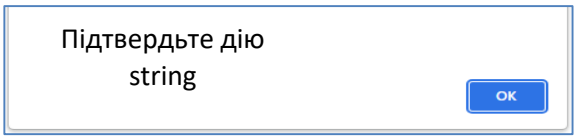
Тип *Object* є особливим типом. Усі інші типи називаються «примітивами», тому що їхні значення можуть містити тільки один елемент (це може бути рядок, число або будь-що інше). В об'єктах же зберігаються колекції даних і більш складні структури, про які буде йтися далі.

***Оператор typeof***

Оператор *typeof* повертає тип аргументу. Це корисно, коли потрібно обробляти значення різних типів по-різному або просто необхідно зробити швидку перевірку.

Виклик *typeof x* повертає рядок із назвою типу. Наприклад, фрагмент дасть такий результат:

```
var name = "Admin";
alert (typeof name);
```




## 5.4 Виведення вікон повідомлень

Для взаємодії з користувачем в браузері існує декілька функцій:

1. **`alert`**() – просте повідомлення. Наприклад (рис. 5.1, а),

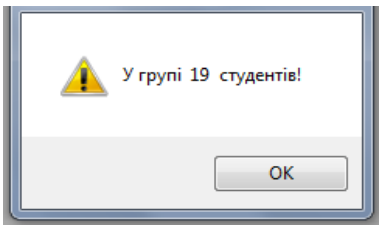
```
var students = 19;
alert ('У групі '+students+' студентів! ');
```

2. **`confirm()`** – повідомлення підтвердження. Наприклад (рис. 4.1, в),

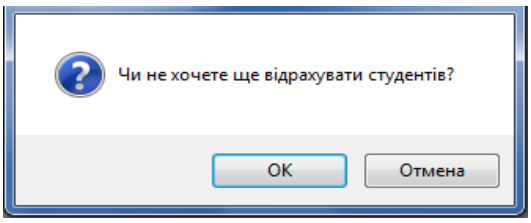
```
if (confirm('Чи не хочете ще відрахувати студентів?'))
     { alert('Студентів більше немає!');}
else { alert(От і добре...');           }
```

3. **`prompt()`** – отримує введення від користувача. Наприклад (рис. 4.1, б),

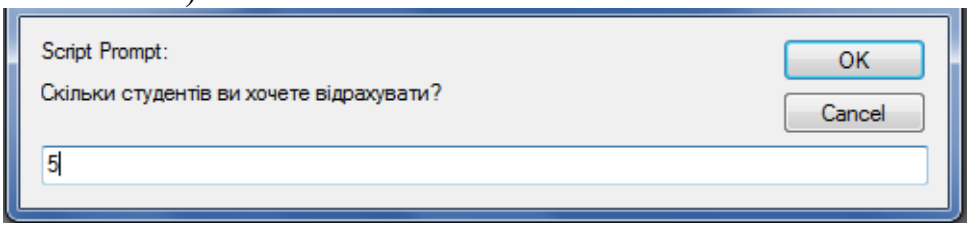
```
var answ = prompt('Скільки студентів ви хочете відрахувати?','1');
```




а)




б)




в)

Рисунок 5.1 – Вигляд вікон повідомлень
(а – *alert*() ; б – *confirm*(); в – *prompt*())

Усі ці методи є модальними: вони призупиняють виконання скриптів та не дають змогу взаємодіяти з рештою сторінки, поки вікно не буде закрите. Є два обмеження, пов'язані з усіма методами вище:

- точне розташування модального вікна визначається браузером (зазвичай у центрі);
- точний вигляд вікна також залежить від браузера. Не можна його змінити.

### 5.5 Основні оператори

Основні оператори мови JavaScript мають таке саме значення і вигляд, як і в інших мовах програмування.

#### 5.5.1 Оператори умови if – else

```
if (умова) {
     // Код, який виконується, коли справедлива умова if
} else {
     // Код, який виконується, коли умова хибна
}
```

Приклад.

```
var answ=prompt('Скільки студентів треба відрахувати?','1');
var xvost = parseInt(answ);
if(xvost > 20){
       alert('Вибачте, але є тільки '+ students +' студентів.');
}
else {
        students -= xvost;
        alert('Залишилося ' + students + ' студентів!');
}
```

Тут використано функцію ***parseInt()***, яка отримує рядок і повертає число.

Крім того, часто для перевірки введення саме числового значення використовується функція ***isNaN()***, яка перевіряє, чи є аргумент числом.

Приклад.

```
var answ=prompt('Скільки студентів треба відрахувати?','1');
var xvost = parseInt(answ);
if (isNaN(xvost)) {
    alert ('Ви повинні ввести допустиме число студентів!');
}
```

#### 5.5.2 Оператори циклів

***while***

```
var i=1;
while (i<=30) {
   document.write (i+'<br />');
   i++;
}
```

***do..while***

```
var i=20;
do {
   document.write('Якщо ви бачите цей текст'+
     'код в циклі був виконаний.');
}
while (i<=3);
```

***for***

```
for (i=1; i<=30; i++) {
   document.write (i+'<br />');
}
```

***for ... in***

```
for (i in car) {
     console.log (car [i] + ' ');
}
```

### 5.5.3 Оператор вибору

Конструкція *switch-case* може замінити декілька *if*, дає можливість більш наочного способу порівняння значення відразу з кількома варіантами.

```
let a = 2;
switch(a) {
     case 3:   alert( 'Замало' );
               break;
     case 4:   alert( 'Точнісінько!' );
               break;
     case 5:   alert( 'Забагато' );
               break;
     default:  alert( 'Я не знаю таких значень' );
}
```

### 5.5.4 Оператори керування

***break***

```
for (i=1;i<=20;i++)
{  document.write(i+'<br />');
   if (i==5)
      break;
}
```

***continue***

```
for (i=1;i<=15;i++) {
   if (i==3) {   continue;   }
   else
     if (i==10) {   continue;  }
     else
          if (i==13) {  continue;   }
   document.write(i+'<br />');
}
```

### 5.6 Функції

Функції – це головні «будівельні блоки» програми. Вони дозволяють робити однакові дії багато разів без повторення коду.

Ми вже стикались з такими вбудованими функціями, як функції виведення вікон повідомлень *alert(), prompt()* та *confirm(),* функції перетворення типу *parseInt()* і функція перевірки на число *isNan().* Але ми також можемо створювати свої функції.

Створення і використання функцій дещо відрізняється від того, як це робиться в інших мовах програмування, хоча суть залишається тією самою:

- для опису функцій використовується ключове слово *function*;
- функція може мати *параметри* і може бути *без параметрів*;
- функція може повертати значення (тоді використовується ключове слово *return*), а може і не повертати. Якщо функція не повертає значення, результат буде *undefined*;
- функції можуть звертатись до *локальних змінних* (оголошених в тілі функції і доступних лише в межах фігурних дужок – тіла функції). Значення, які передаються в функцію як параметри, копіюються саме в локальні змінні;
- функції мають доступ до *зовнішніх змінних* (глобальних відносно функції). І тоді їх не потрібно передавати через параметри. Але це працює тільки зсередини назовні. Код поза функцією не має доступу до локальних змінних функції);
- *виклик функції* – це вказання її імені і переліку фактичних параметрів, якщо вони є.

Приклади.

| *Опис функції* | *Пояснення* | *Виклик функції* |
|---|---|---|
| `function showMessage() {`<br>`  alert('Всім привіт!');`<br>`}` | Функція без параметрів, не повертає нічого | `showMessage();` |
| `function showMessage(from, text){`<br>`  alert(from + ': ' + text);`<br>`}` | Функція з двома параметрами, не повертає нічого | `showMessage(`<br>`  "Login",`<br>`  "admin");` |
| `function checkAge(age) {`<br>`  if (age >= 18) {`<br>`  return true;`<br>`  } else {`<br>`  return confirm('Ви-студент?');`<br>`  }`<br>`}` | Функція з одним параметром, повертає значення | `checkAge(22);` |

У мовах Java, C++, C# та інших останнім часом для простоти опису і кращого розуміння функціонального підходу використовують анонімні функції. JavaScript також може використовувати ***анонімні функції***.

Приклади.

```
var printText = function (a) {document.write (a);};
   // Тут якщо функція повертає результат, він буде у змінній printText
var printText = function (a)
        {document.write (a);} ("Hello World!");
   // Функція відразу може бути викликана з фактичним параметром
var printText = (function (a)
        {document.write (a);} ("Hello World!"));
   // Рекомендується брати в дужки – не обов'язково, але зрозуміліше
```

### 5.7 Виключні ситуації

Інструмент виключних ситуацій дуже популярний серед програмістів й існує він в усіх мовах програмування.

Спочатку про основні поняття, пов'язані з виключними (винятковими) ситуаціями:

- ***Виключення*** (виняток) – це подія, яка сигналізує про ненормальну ситуацію чи помилку.
- ***Генерувати виключення*** (створювати) – сигналізувати про будь-яку помилку чи виняткову ситуацію. Виключення генерується оператором «*throw*»,
- ***Ловити виключення*** – це вжити заходів для обробки виключення та відновлення нормальної функціональності коду.
- ***Перехоплення виключення*** реалізується оголошенням блока «*try-catch-finally*».

Приклад.

```
function testFactorial (inputData) {
     if (inputData <0)
            throw "Число не повинно бути менше нуля";
     return (inputData - 1)?
            (inputData*testFactorial(inputData-1)):
             inputData;
}
...
var myNumber = -5;
try{
   document.write (testFactorial (myNumber));
} catch(ex) {
   document.write (ex);
}
```

У цьому прикладі функція `testFactorial()` генерує (*throw*) виключну ситуацію у випадку від'ємного числа. Якщо під час виклику цієї функції як фактичний параметр попадає від'ємне число, ця ситуація «відловлюється» у блоці *try-catch*.

За виникнення стандартного виключення можна створити екземпляр об'єкта ***Error***.

Текст повідомлення записується у властивість *message* об'єкта, а назва виключення знаходиться у полі *name*.

Приклад.

```
try {
     throw new Error('Something went wrong!');
}
catch (e) {
     console.log (e.name + ':' + e.message);
}
```

### 5.8 Об’єкти мови JavaScript

Ми знаємо, що у JavaScript є вісім типів даних. Сім з них називаються «примітивними», оскільки їх значення містять лише одну річ (чи то рядок, число чи щось інше). На противагу цьому, об’єкти використовуються для зберігання складніших об’єктів.

JavaScript – об'єктно-орієнтована мова програмування дозволяє визначати наші власні об'єкти і створювати наші власні типи даних. І, крім того, JavaScript надає широкий спектр готових об’єктів.

У JavaScript об’єкти проникають майже в усі аспекти мови (рис. 5.2).

Як бачимо з рисунка, є три види об’єктів:

- об’єкти ядра мови (***JavaScript Core***);
- об’єкти документа (***DOM***);
- об’єкти браузера( ***BOM***).

Кожен об’єкт подано певним класом, тобто, кожен об’єкт має властивості і методи. Властивості об'єкта (поля) – це деякі значення, пов'язані з цим об'єктом. Наприклад, для об'єкта *String* є властивість *length*:

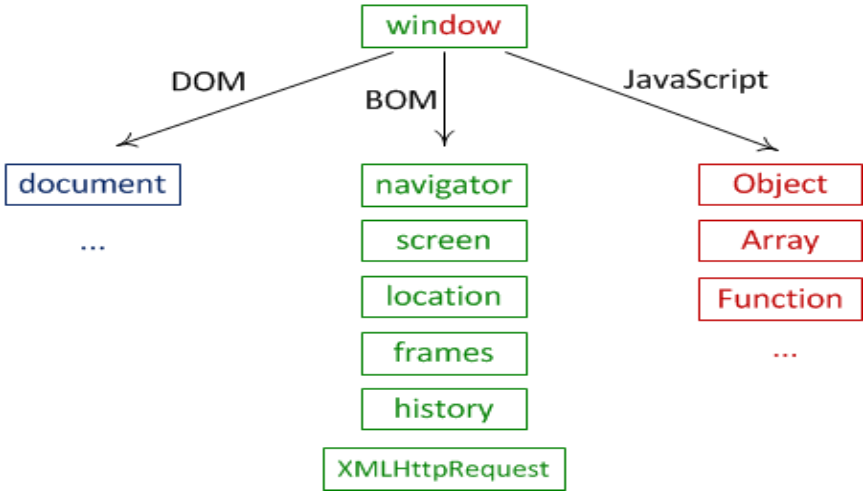



Рисунок 5.2 – Об’єктна модель JavaScript

```
var  textString = "Hello world!";
console.log(textString.length);
```

Методи – це дії, які можна виконати над об'єктом, наприклад,

```
var textString = "Hello world!";
console.log(textString.toUpperCase());
```

### 5.9 Створення об’єкта і маніпуляції з ним

Власний об’єкт можна створити двома способами.

1. Створити *прямий екземпляр* об'єкта:

```
var  person = new Object ();
```

або

```
var  person = { }
```

2. Створити об'єкт *за допомогою конструктора*, в якому одразу вводяться назви властивостей об’єкта і здійснюється їх ініціалізація:

```
function Person (name, age, year) {
     this.name = name;
     this.age = age;
     this.year = year;
}
```

Тоді створення екземплярів об'єкта буде таким:

```
var employee1 = new Person ("Ivan", "25", "2017");
var employee2 = new Person ("Olga", "21", "2016");
```

Далі за необхідності можна *додати нову властивість* об'єкта:

```
person.name = "Ivan"
```

або

```
person ['name'] = "Ivan"
```

або (за створення об'єкта):

```
var  person = { name: "Ivan", age: 25  }
```

*Доступ до властивості* може бути виконаний таким чином:

```
console.log (person.name);
```

або

```
console.log (person ['name']);
```

*Видалити властивість* за необхідності можна так:

```
delete person.name;
```

Можна *додавати методи* до об'єкта в процесі роботи:

```
var person = {}
person.sayAge = function(n) {
     console.log ("Person is" + n + "years old");
};
```

!!! Додання методу до об'єкта фактично присвоює функцію деякій властивості об'єкта.

Для доступу до властивостей об'єкта з методу використовують ключове слово ***this***:

```
person.sayName =  function () {
                console.log ("My name is" + this.name);
            }
```

За потреби можна ***переглянути усі властивості*** об'єкта. Наприклад, якщо маємо об’єкт, описаний таким чином:

```
var person = {
     name: "Ivan",
     age: 25,
     hiredYear: 2017
}
```

то для перегляду його усіх властивостей можна використати функцію:

```
person.sayAll = function() {
     for (var i in this) {
          console.log(i + " is " + this[i]);
      }
}
```

### 5.10 Об’єкти ядра мови JavaScript

Розглянемо деякі з вбудованих об’єктів мови (рис. 5.3).

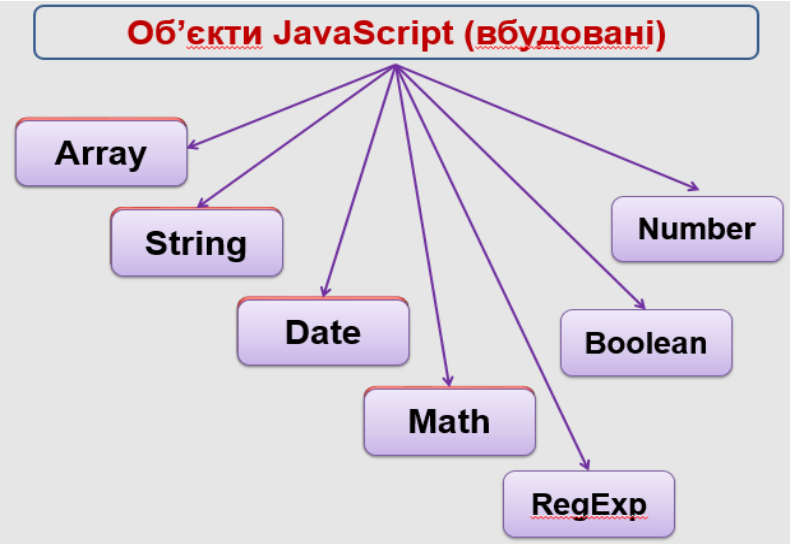



Рисунок 5.3 – Основні об’єкти мови

#### 5.10.1 String (рядки)

Створення об’єкта-рядка:

```
myString = 'my string';
myString = String(object);
```

Властивість: **`length`.**

Рядок – це масив символів. Тому доступ до символів – це доступ до елементів масиву (нумерація елементів масиву – з 0). Наприклад,

```
var user = "User";
console.log(user[1]);
```

Оскільки робота з рядками дуже необхідна у web-програмуванні, то цей об’єкт має багато зручних методів.

Основні методи класу String зведено у табл. 5.1.

Таблиця 5.1 – Методи об’єкта String

| *Назва методу* | *Призначення* |
|---|---|
| **charAt(n)** | Повертає символ з індексом n |
| **concat**(str1,str2,...,strN) | Об’єднує рядки в один і повертає його |
| **toLowerCase**()<br>**toUpperCase**()<br>**toLocaleLowerCase**()<br>**toLocaleUpperCase**() | Приведення рядка до верхнього (нижнього) регістра |
| **split**('separator') | Розділення рядка вказаним символом |
| **fromCharCode**(num1, num2, ..., numN) | Створює елементарний рядок зі значень символів Unicode |
| **toString**() | Повертає елементарний рядок замість об'єкта String |
| **valueOf**() | Повертає елементарний рядок замість об'єкта String, еквівалентний toString () |
| **substring**(indexA [, indexB]) | Повертає частину рядка, починаючи з позиції indexA, але не включає indexB |
| **slice**(indexA [, indexB]) | Повертає частину рядка, починаючи з позиції indexA, але не включає indexB |
| **substr**(indexA [, length]) | Повертає частину рядка, починаючи з позиції indexA, довжиною length |
| **replace**(regexp, newSubString\|function) | Заміна у регулярному виразі |
| **indexOf**(searchValue[, fromIndex]) | Шукає підрядок searchValue, починаючи з позиції fromIndex |
| **lastIndexOf**(searchValue[, fromIndex]) | Шукає останній підрядок searchValue, починаючи з fromIndex |
| **search**(regExp) | Перевіряє, чи є відповідність аргументу, результат true або false |
| **match**(regExp) | Повертає масив усіх збігів з аргументом |
| *Методи обгортки (заключити рядок між тегами)* | |
| **bold()** | Обгортає рядок тегами <b> |
| **italics()** | Обгортає рядок тегами <i> |
| **link()** | Обгортає рядок тегами <a> |
| **sub()** | Обгортає рядок тегами <sub> |
| **sup()** | Обгортає рядок тегами <sup> |

### 5.10.2 Array (масиви)

Масиви в JavaScript – ***це об'єкти***!

Масиви в JavaScript ***нетипізовані*** – дозволяють зберігати елементи різних типів всередині одного масиву. Це можуть бути не тільки елементарні типи – рядки, числа чи символи, але й об’єкти та масиви і навіть масиви масивів.

Масиви в JavaScript є ***динамічними***, тобто вони можуть збільшуватися або зменшуватися в розмірах. Оголошувати заздалегідь фіксовані розміри

під час їх створення, а також перерозподіляти пам'ять під час зміні розміру, як це робиться в деяких інших мовах, не обов'язково.

Існує декілька способів створення масивів в JavaScript.

1. За допомогою конструктора

```
item = new Array();
item[0] = "AES";
item[1] = "RES";
item[2] = "SHA2";
item[3] = "MD5";
```

або

```
var myArray = new Array (10);
```

2. Конструктор з ініціалізацією

```
var myArray = new Array("Ivanov","Petrov","Sidorov", "Kuzsov");
```

3. Ініціалізація («буквально»)

```
var myArray = ["Ivanov", "Petrov", "Sidorov", "Kuznetsov"];
```

Методи об’єкта *Array* наведено у таблиці 5.2.

Таблиця 5.2 – Методи об’єкта Array

| *Назва методу* | *Призначення* |
|---|---|
| **push**( ) | додання елемента в кінець масиву |
| **unshift**( ) | додання елемента в початок масиву |
| **concat**(element) | додає до масиву ще один елемент |
| **delete**<br>напр.,<br>delete arr [2]; | видалення елемента з масиву (значення 3-го елемента буде не визначене, але кількість елементів у масиві не змінюється, всі інші елементи залишаються на своїх місцях) |
| **pop**() | видаляє елемент в кінці масиву, зменшує довжину масиву на 1 і повертає значення видаленого елемента |
| **shift**(); | видаляє елемент на початку масиву і зміщує всі елементи на 1 позицію вліво |
| **join**('s') | повертає рядок – всі елементи масиву, з’єднані символом *s* |
| **reverse**() | повертає масив, у якому елементи знаходитимуться у зворотному порядку |
| **sort**() | сортує масив у природному порядку |
| **splice**(arg1,arg2,<br>arg3, …) | метод змінює початковий масив, але повертає масив видалених елементів. Якщо жоден елемент не буде видалений, повертається порожнє значення.<br>*arg1* – це позиція елемента, з якого починається дія методу<br>*arg2* – кількість елементів, які потрібно видалити, починаючи з arg1 (якщо arg2 не вказано, будуть видалені всі елементи, починаючи з arg1).<br>*arg3* і наступний (будь-яка кількість) – це елементи масиву, які будуть додані, починаючи з позиції, зазначеної в першому аргументі |
| **slice**(k,n) | повертає підмасив з масиву, від *k*-ого до *n*-ого елемента (але не включаючи *n*-ий) |

### 5.10.3 Date (дата)

Цей об’єкт призначений для роботи з датою і часом.
Конструктори:

```
var x = new Date()
var x = new Date(milliseconds)
var x = new Date(stringData)
var x = new Date(year, month, day [, hours, min, sec, msec)
```

Приклад

```
var myDate = new Date ()
      // Mon May 15 2017 19:20:25 GMT + 0300 (RTZ 2 (winter))
var myDate = new Date ("December 14, 1975 12:10:00")
      // Sun Dec 14 1975 12:10:00 GMT + 0300 (RTZ 2 (winter))
var myDate = new Date (1989, 6, 14)
      // Fri Jul 14 1989 00:00:00 GMT + 0400 (RTZ 2 (summer))
var myDate = new Date (1998, 6, 14, 11, 20, 00)
      // Tue Jul 14 1998 11:20:00 GMT + 0400 (RTZ 2 (summer))
```

Всі методи об’єкта Date розіб’ємо на декілька груп за їх призначенням і для розуміння їх роботи наведемо приклади.

*Встановлення, зміна дати*

```
var myDate = new Date ();
myDate.setFullYear (2017, 4, 22);
myDate.setDate (myDate.getDate () + 10);
```

*Порівняння дат*

```
var currentDate = new Date ();
var nextNewYear = new Date ();
nextNewYear.setFullYear (2018, 0, 1);
if (currentDate == nextNewYear) {
     alert ("New 2018th Year! Ura! !!!");
}
```

*Отримання інформації про дату*

**`getDay()`** – повертає день тижня від 0 до 6, 0 – неділя, 1 – понеділок тощо;
**`getTimeZoneOffset()`** – повертає зміщення часового поясу відносно UTC, у хвилинах із протилежним знаком;
**`getYear()`** – повертає значення року мінус 1900, користуватися не рекомендується;
**`getFullYear()`** – повертає значення року;
**`getMonth()`** – повертає місяць, від 0 до 11;
**`getDate()`** – повертає день місяця з 1 по 31;
**`getHours()`** – повертає годину, від 0 до 23;
**`getMinutes()`** – повертає кількість хвилин, від 0 до 59;
**`getSeconds()`** – повертає кількість секунд від 0 до 59;
**`getMilliseconds()`** – повертає кількість мілісекунд, від 0 до 999;
**`getTime()`** – повертає кількість мілісекунд, що минули з 1 січня 1970.

*Встановлення інформації в дати*

**setYear()** – встановлює значення року мінус 1900, не рекомендується для використання;

**setFullYear()** – встановлює значення року;

**setMonth()** – встановлює місяць, від 0 до 11;

**setDate()** – встановлює день місяця від 1 до 31;

**setHours()** – встановлює годину, від 0 до 23;

**setMinutes()** – встановлює кількість хвилин від 0 до 59;

**setSeconds()** – встановлює кількість секунд від 0 до 59;

**setMilliseconds()** – встановлює кількість мілісекунд, від 0 до 999;

**setTime()** – встановлює кількість мілісекунд, що минуло з 1.01.1970 р.

*Перетворення дати*

**Date.parse()** – перетворює рядок з датою, наприклад «05 липня 2017 року» і повертає кількість мілісекунд, що минули з 1.01.1970 року. Якщо рядок не вдалося перетворити, метод повертає NaN;

**toLocaleString()** – повертає об'єкт String – дату у довгому форматі, наприклад «10 січня 2017 12:26:01» відповідно до регіональних налаштувань операційної системи, в якій запущено сценарій. Зовні різниця буде, наприклад, між країнами, де 12-годинне позначення часу є загальним, порівняно з 24-годинним часом;

**toLocaleTimeString()** – перетворює дані про час у рядок, використовуючи параметри форматування операційної системи, в якій виконується сценарій;

**toLocaleDateString()** – виконує перетворення, подібне до попереднього, але з датою.

### 5.10.4 Math (математичні функції)

Цей об'єкт призначений для виконання різних математичних функцій. Особливість його в тому, що:

- не має конструкторів;
- властивості та методи – статичні.

Це означає, що можна отримати доступ до його методів та властивостей безпосередньо, не створюючи екземпляр об'єкта.

Наприклад,

```
PI ~ Math.PI
```

або

```
sqrt(x) ~ Math.sqrt(x).
```

*Константи*

**Math.E** – число e, основа природного логарифма, константа Ейлера (Непера), приблизно 2,718 ...;

**Math.PI** – число Pi, що приблизно дорівнює, як відомо, 3,1415926 ...;

**Math.SQRT2** – квадратний корінь 2, приблизне значення – 1,414;
**Math.SQRT1_2** – квадратний корінь 1/2, приблизне значення – 0,707;
**Math.LN2** – природний логарифм 2, приблизне значення – 0,693;
**Math.LN10** – природний логарифм 10, приблизне значення 2,302;
**Math.LOG2E** – логарифм Е бази 2, приблизне значення 1,444;
**Math.LOG10E** – логарифм Е бази 10, приблизне значення 0,434.

*Тригонометричні функції*

**Math.sin(x)** – повертає синус аргументу (в радіанах) від -1 до 1;
**Math.cos(x)** – повертає косинус аргументу (в радіанах) від -1 до 1;
**Math.tan(x)** – повертає числове значення дотичної точки кута в радіанах;
**Math.asin(x)** – повертає значення (рад.) арксинуса аргументу, від -1 до 1
**Math.acos(x)** – повертає значення (рад.) арккосинуса, від -1 до 1;
**Math.atan(x)** – повертає значення арктангенса (від *–pi/2* до *pi/2*) аргументу;
**Math.atan2(x,y)** – функція називається арктангенсом двох змінних, повертає числове значення між *-pi* та *pi* і являє собою кут між додатною віссю X і точкою (*x*,*y*).

*Функції порівняння і перетворення*

**Math.min([Value1[,value2[,...]]])** – повертає мінімальне значення аргументів;
**Math.max([Value1[,value2[,...]]])** – повертає максимальне значення аргументів;
**Math.floor(x)** – повертає найбільше ціле число, що менше або дорівнює аргументу;
**Math.ceil(x)** – повертає найменше ціле число, що перевищує або дорівнює аргументу;
**Math.abs(x)** – повертає абсолютне значення числа, його також називають «модулем»;
**Math.round(x)** – округлює число за правилами математики.

*Функції обчислення*

**Math.sqrt(x)** – повертає квадратний корінь аргументу;
**Math.pow(base,exp)** – підносить число «base» до степеня «*exp*»;
**Math.log(x)** – обчислює натуральний (на основі *e*) логарифм числа;
**Math.exp(x)** – обчислює показник – значення числа е в степені «х»;
**Math.random()** – повертає випадкове число від 0 (включно) до 1.

### Контрольні запитання

1. Яка різниця між мовою JavaScript та іншими мовами програмування?
2. Які способи додання скриптів на html-сторінку існують?
3. Які види коментарів існують у мові JavaScript?
4. Якими можуть бути ідентифікатори у мові JavaScript?
5. Що ви можете сказати про типи даних в JavaScript?

6. Як ввести змінну у код скрипта?
7. Які базові типи даних має мова JavaScript?
8. Які вікна повідомлень є в JavaScript і які функції їх реалізують?
9. Назвіть основні види операцій і наведіть приклади.
10. Яка різниця між локальними і глобальними змінними?
11. Яка різниця в термінах дії змінних, оголошених за допомогою *let* і *var*?
12. Наведіть приклади основних видів операторів у мові JavaScript.
13. Як реалізують функції у мові JavaScript?
14. Що таке анонімні функції? Наведіть приклади.
15. Як обробляються виключні ситуації у мові JavaScript?
16. Які види об'єктів наявні в мові JavaScript?
17. Назвіть основні об'єкти ядра мови і охарактеризуйте їх.

## Практична робота № 5

### *Мета роботи*

- Ознайомитись з базовими поняттями і можливостями мови JavaScript та отримати навички у застосуванні основних конструкцій мови.
- Ознайомитись з об'єктами ядра мови JavaScript.
- Навчитись розробляти і на практиці використовувати власні скрипти у web-сторінках, виконавши індивідуальне завдання.

### *Завдання і порядок виконання роботи*

Використовуючи мову JavaScript, доповнити розроблену web-сторінку, додавши в неї розділ (пункт меню) «Практична робота з JavaScript». На сторінці для кожної задачі розмістити інформацію:

- номер варіанта і умову задачі (реалізувати окремою функцією);
- початкові дані (дані можуть вводитись за допомогою *prompt*, але на сторінці після введення з'являтись у вигляді тексту*)*;
- результати виконання задачі.

*Задача 1.* Необхідно розробити вказану функцію і зробити так, що програма циклічно виконувалась до тих пір, поки користувач не введе певне значення для закінчення обчислень.

| | |
|---|---|
| 1 | Напишіть програму, яка вводить довільні натуральні числа і визначає, чи є введене число паліндромом. Для визначення, чи є число паліндромом, написати окрему функцію, яка приймає число і повертає true або false. Число називається паліндромом, якщо воно зліва направо і справа наліво читається однаково. Наприклад, числа 7667 і 34543 є паліндромами, а 123 і 45 – ні. |
| 2 | Напишіть функцію, яка отримує число і друкує це число як послідовність цифр, між якими стоять два пробіли. Наприклад, ціле число 4562 має бути надруковане так: «4 5 6 2». Використати цю функцію для друку чисел, згенерованих на проміжку (1000, 32000). |

<table>
<tr><td>3</td><td>Напишіть функцію, яка отримує час у вигляді трьох цілих аргументів (години, хвилини, секунди) і повертає кількість секунд з моменту, коли на годиннику було 0 годин. Використайте цю функцію для підрахунку часу в секундах між двома моментами часу.</td></tr>
<tr><td>4</td><td>Напишіть функцію, яка виводить ціле число між 1та 32767 і друкує це число як послідовність цифр, між якими стоять два пробіли. Наприклад, ціле число 4562 має бути надруковано так: 4 5 6 2.</td></tr>
<tr><td>5</td><td>Ціле число називається досконалим (рос. совершенным), якщо сума його дільників, включно 1 (але не саме число) , дорівнює цьому числу. Наприклад, число 6 = 1+2+3 є досконалим. Напишіть функцію perfect, яка визначає, чи є число досконалим. Використайте цю функцію у програмі, яка знаходить і друкує усі досконалі числа з діапазону від 1 до 1000.</td></tr>
<tr><td>6</td><td>Ціле число називається простим, якщо воно ділиться на 1 і на самого себе. Наприклад, числа 2, 3, 5 і 7 є простими, а 2, 6, 8 і 9 – ні. Напишіть функцію, яка визначає, чи є число простим. Використайте цю функцію у програмі, яка знаходить і друкує усі прості числа у діапазоні від 1 до 10000.</td></tr>
<tr><td>7</td><td>Напишіть функцію, що отримує ціле значення і повертає число з оберненим порядком цифр. Наприклад, для 7631 функція має повернути 1367. Використайте цю функцію для виведення будь-якого числа, введеного користувачем, у реверсному вигляді.</td></tr>
<tr><td>8</td><td>Напишіть функцію, яка отримує бали, зароблені студентом, а повертає оцінку за шкалою ECTS:
<table>
<tr><td>Сума балів</td><td>90 - 100</td><td>82-89</td><td>74-81</td><td>64-73</td><td>60-63</td><td>35-59</td><td>0-34</td></tr>
<tr><td>Оцінка ECTS</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>FX</td><td>F</td></tr>
</table>
Використайте цю функцію для отримання оцінки за введеними балами.</td></tr>
<tr><td>9</td><td>Напишіть функцію distance, яка обраховує відстань між двома точками з координатами (x1, y1) і (x2, y2). Усі числа і повернені значення мають бути дійсними. Використайте цю функцію у програмі, яка обчислює площу трикутника за формулою Герона.</td></tr>
<tr><td>10</td><td>Напишіть програму, яка допоможе студенту вивчити таблицю множення. Згенеруйте два додатних однорозрядних числа (окрема функція). Програма виводить, наприклад, питання: «Скільки буде 5 на 6?». Студент відповідає. Якщо відповідь правильна, програма друкує: «Молодець! Дуже добре!». І далі задає наступне питання на множення. Якщо відповідь неправильна, програма друкує: «Неправильно! Спробуйте знову …», до тих пір, поки відповідь не буде правильною. Розроблювана функція має отримувати три числа: два множники та число – відповідь студента, і друкувати потрібну фразу.</td></tr>
<tr><td>11</td><td>Напишіть функцію, яка отримує натуральне число n, а повертає випадкове число, що складається з n цифр. Використайте цю функцію для генерування числа довжиною в n цифр (n задається користувачем).</td></tr>
</table>

| | |
|---|---|
| 12 | Напишіть програму, яка грає у гру «Вгадай число» таким чином: ваша програма «задумує» число (випадкове число у діапазоні від 1 до 1000), яке потрібно вгадати. Далі програма друкує:<br>*У мене є число між 1 та 1000. Відгадайте і введіть ваше число…*<br>Далі гравець вводить перше число. Програма відповідає однією з фраз:<br>*Чудово! Ви вгадали число! Будете грати далі?*<br>*Занадто мале. Спробуйте ще раз.*<br>*Занадто велике. Спробуйте ще раз.*<br>Під час реалізації гри необхідно написати функцію, яка набуває двох чисел: «задумане» і відповідь гравця, а після аналізу друкувати одну з фраз. |
| 13 | Написати функцію, яка отримує 4 цифри, а повертає ціле десяткове число. Наприклад, введено числа 3, 7, 5, 9, а отримане число – 3759. Використати цю функцію для формування числа з цифр, введених користувачем. |
| 14 | Напишіть програму, яка буде використовувати функцію, що обчислює скалярний добуток двох векторів дійсних чисел однакової розмірності. |
| 15 | Розробити функцію, яка змінює значення двох заданих дійсних змінних. Перша змінна має отримати значення суми початкових даних, а друга – значення їх різниці. |
| 16 | Напишіть функцію *muliply()* для двох цілих, яка визначатиме, чи кратне друге число першому. Функція має отримувати два цілих аргументи і повертати 1 (*true*), якщо друге число кратне першому, і 0 (*false*) – в іншому випадку. Використайте цю функцію у програмі, яка вводить серію пар цілих чисел. |

*Задача 2.* Засобами JavaScript створити масив (розмір матриці вводити через вікна повідомлень), елементи якого – випадкові числа на проміжку (*a, b)*, і виконати дії, вказані в індивідуальному завданні.

– Вхідні дані (розмір, інтервал) перевіряти на правильність введення і виводити на сторінку;
– Вхідний і результуючий масиви вивести на сторінці у зручному вигляді, виділивши іншим кольором у результуючій матриці змінені елементи.

| | |
|---|---|
| 1 | Реалізувати програму, яка міняє місцями перший і останній стовпці матриці дійсних чисел. |
| 2 | Реалізувати програму, яка додає перший і останній рядки цілочисельної матриці і записує результат у останній стовпець. |
| 3 | Реалізувати програму, яка міняє значення елементів квадратної матриці на значення відповідних елементів заданого одновимірного масиву. |
| 4 | Реалізувати програму, яка додає відповідні елементи двох заданих масивів цілих чисел і заносить результат у третій масив. |
| 5 | Реалізувати програму, яка міняє місцями перший рядок і останній стовпець матриці дійсних чисел. |
| 6 | Реалізувати програму, яка підсумовує елементи рядків двовимірного масиву і заносить результат в одновимірний масив, розмірність якого дорівнює числу рядків двовимірного масиву. |

| | |
|---|---|
| 7 | Реалізувати програму, яка міняє місцями діагоналі цілочисельної матриці. |
| 8 | Реалізувати програму, яка знаходить максимальний за модулем елемент заданого двовимірного масиву дійсних чисел. |
| 9 | Реалізувати програму, яка міняє місцями останній рядок і перший стовпець цілочисельної матриці. |
| 10 | Реалізувати програму, яка додає перший і останній стовпці цілочисельної матриці і записує результат на місце першого рядка. |
| 11 | Реалізувати програму, яка міняє елементи заданого стовпця матриці дійсних чисел на значення відповідних елементів одновимірного масиву. |
| 12 | Реалізувати програму, яка перемножає відповідні елементи двох заданих матриць і заносить результат у третю матрицю. |
| 13 | Реалізувати програму, яка міняє місцями останній рядок і перший стовпець цілочисельної матриці. |
| 14 | Реалізувати програму, яка підсумовує елементи стовпців двовимірного масиву і заносить результат в одновимірний масив, розмірність якого дорівнює числу стовпців двовимірного масиву. |
| 15 | Реалізувати програму, яка знаходить номер рядка заданого двовимірного масиву, що має максимальну за модулем суму елементів. |

*Задача 3.* Знайти в мережі Інтернет цікаві готові скрипти (функції) мовою JavaScript, завантажити їх у свій web-проєкт і використати на своїх сторінках.

**Додаткові джерела**


1. Сучасний підручник з javascript. [Електронний ресурс]: URL: https://uk.javascript.info/.
2. JavaScript : Підручник. Основи веб-програмування. [Електронний ресурс]: URL: https://w3schoolsua.github.io/js/index.html#gsc.tab=0.
3. Мельник Р. А. Програмування веб-застосувань (фронтенд та бекенд). Львів : Львівська політехніка, 2018. 248 с.
4. Бородкіна І. Л., Бородкін Г. О. WEB-технології та WEB-дизайн: застосування мови HTML для створення електронних ресурсів. К. : Ліра-К, 2020. 212 с.
5. Руденко В. Д., Речич Н. В., Потієнко В. О. Інформатика. Профільний рівень. Харків: Ранок, 2020. 256 с.
6. JavaScript Cookbook: Programming the Web 3rd Edition. Adam D. Scott, Matthew MacDonald, Shelley Powers. O'Reilly Media, Inc..August 24, 2021. ISBN: 9781492055709.
7. PC VECTOR. [Електронний ресурс]: URL: https://pcvector.net/.
8. Об'єктно-орієнтоване програмування. [Електронний ресурс]: URL: http://xn--80adth0aefm3i.xn--j1amh/%D0%9E%D0%9E%D0%9F.
9. Стандартні вбудовані об'єкти. [Електронний ресурс]: URL: https://webdoky.org/uk/docs/Web/JavaScript/Reference/Global_Objects/#standartni-obiekty-za-katehoriiamy.

# 6 ОБ'ЄКТНА МОДЕЛЬ ДОКУМЕНТА (DOM). ЕЛЕМЕНТИ КЕРУВАННЯ І ФОРМИ

## 6.1 Об'єктна модель документа

Об'єктна модель документа – це дещо інше подання веб-сторінки, ніж html-код. Браузер за вказаною URL-адресою відправляє запит і отримує (завантажує) з сервера веб-сторінку у вигляді html-коду, який часто називається вихідний код сторінки. І якщо у коді вказано інші файли, такі як стилі css, js – то завантажує і їх. Далі, із завантаженого з сервера html-коду браузер формує DOM – Document Object Model. Це робиться для того, щоб за допомогою JavaScript можна було швидко маніпулювати веб-документом: шукати потрібний елемент, додавати нові елементи, отримувати наступний дочірній елемент і т. д.

Вигляд html-коду і ієрархії елементів документа наведено на рис. 6.1.

```
<html>
<head>
    <meta charset="UTF-8">
    <title> приклад </title>
    <link  rel="stylesheet"
      type="text/css"href="stl.css"/>
</head>
<body >
  <div class="wrapper">
     <div class="content"></div>
        <header class="header">
             HEADER  </header>
        <p> Логін
        <INPUT NAME=login SIZE=10>
            Пароль
        <INPUT TYPE=password NAME="psw">
        </p>
     </div>
     <div class="footer">
            Footer
     </div>
   </div>
</body>
</html>
```

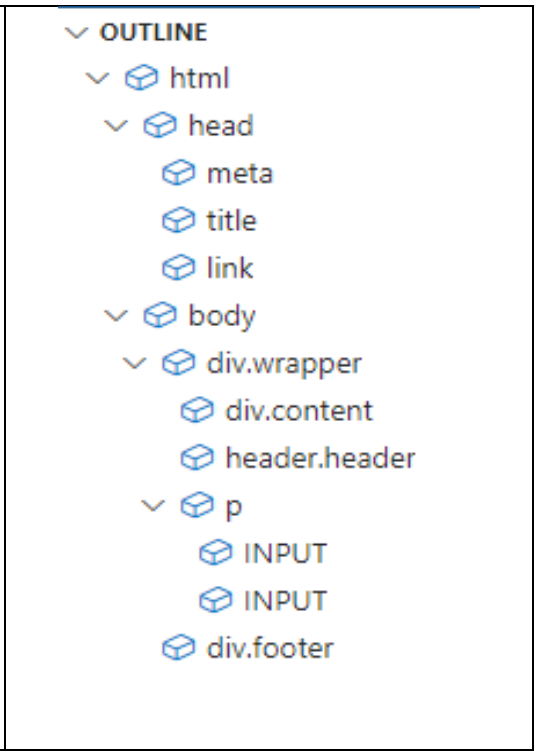



Рисунок 6.1 – Приклад ієрархії елементів документа

### 6.1.1 Вузли об'єктної моделі документа

Отже, під час завантаження сторінки створюється об'єкт *window*, що має властивість *document*, для якого будується ієрархія об'єктів документа (рис. 6.2), а браузер надає доступ до неї.

Об'єктна модель документа складається з вузлів. Вузли бувають:

- *елементними* (на рисунку вони позначені зеленим кольором),
- *текстовими* (блакитним кольором);
- *атрибутними* (рожевий колір) – для кожного атрибута html-елемента.

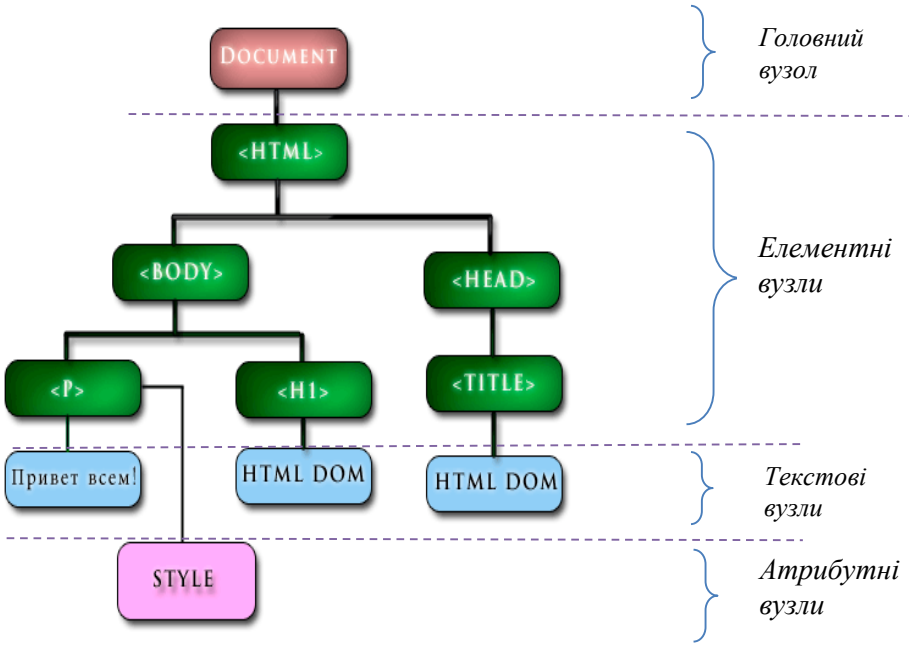



Рисунок 6.2 – Приклад об’єктної моделі документа

***Кожен вузол – це об’єкт***, а отже, має властивості і методи. Між об’єктами існують певні відносини:

- *кореневий* вузол – самий верхній вузол в DOM (<html>);
- *батьківський* вузол (parent node);
- *вузли-нащадки* (child node);
- *вузли-брати* (sibling node).

!!! Об'єкт *document* не є частиною *DOM*. *Document* містить *DOM*.

### 6.1.2 Звернення до елементів DOM

Для звернення до елементів DOM можна використати два способи:

1. Послідовне переміщення (складно, не завжди зрозуміло).
2. Пряме звернення (зручно, зрозуміло)

Наприклад, для доступу до атрибутного вузла Style, що зображено на рис. 6.2, необхідно побудувати такий ланцюжок виразів:

```
document.body.childNodes[0].ChildNodes[0].NodeValue
```

або

```
document.documentElement.childNodes[0].ChildNodes[0].NodeValue.
```

## 6.2 Робота з DOM

### 6.2.1 Пошук елементів

**`getElementById(value)`**– вибирає елемент, у якого атрибут *id* дорівнює *value*. Якщо елемента з таким ідентифікатором немає, то повертається *null;*

**`getElementsByTagName(value)`** – вибирає всі елементи, у яких значення тегу дорівнює *value* (повертає масив елементів);

**`getElementsByName(name)`** – вибирає всі елементи, у яких значення тегу дорівнює *name* (повертає масив елементів);

**`getElementsByClassName(value)`** – вибирає всі елементи, які мають клас *value* (повертає масив елементів);

**`querySelector(value)`** – вибирає перший елемент, який відповідає css-селектору *value;*

**`querySelectorAll(value)`** – вибирає всі елементи, які відповідають css-селектору *value.*

*Приклади.* Далі наводиться лише частина коду (зліва) і вигляд результуючої сторінки (справа).

1. За допомогою виклику *document.getElementById("header")* знаходимо елемент, у якого *id="header"*. За допомогою властивості *innerText* можна отримати текст знайденого елемента.

```
<body>
    <h3 id="header">Деякий заголовок</h3>
    <p>Це текст абзаца</p>
<script>
    var headEl =
document.getElementById("header");
    document.write("Текст заголовка: " +
                   headEl.innerText);
</script>
</body>
```

**Деякий заголовок**

Це текст абзаца

Текст заголовка: Деякий заголовок

2. За допомогою виклику *document.getElementsByTagName("p")* знаходимо всі елементи параграфів. Цей виклик повертає масив знайдених елементів. Для отримання окремих елементів необхідно пройтися по них у циклі.

```
<body>  <h3>Заголовок</h3>
        <p>Перший абзац</p>
        <p>Другий абзац</p>   <hr>
<script>
var Elements=document.getElementsByTagName("p");
for (var i = 0; i < pElements.length; i++) {
    document.write("Текст параграфа: "
            + pElements[i].innerText+"<br/>");
}
</script>
</body>
```

**Заголовок**

Перший абзац

Другий абзац

Текст параграфа: Перший абзац
Текст параграфа: Другий абзац

3. Доступ до елементів за класом.

```
    ...
    <body> <div class="article">
           <h3>Заголовок</h3>
           <p class="text">Перший абзац</p>
           <p class="text">Другий абзац</p> <hr>
```

**Заголовок**

Перший абзац

Другий абзац

Клас: [object HTMLDivElement]
Елемент: Перший абзац
Елемент: Другий абзац

```
    </div>
<script>
var articleDiv = document.getElementsByClassName("article")[0];
document.write("Клас: "+articleDiv+"<br/>");
var textElems = document.getElementsByClassName("text");
for (var i = 0; i < textElems.length; i++) {
    document.write("Елемент: "
            + textElems[i].innerHTML+"<br/>");
}
</script>
</body>
```

4. Доступ до елементів за селектором

```
<body>
     <div class="annotation">
        <p>Анотація статті</p>
      </div>
    <div class="text">
        <p>Перший абзац</p>
        <p>Другий абзац</p>
    </div> <hr>
<script>
var elem = document.querySelector(".annotation p");
document.write("Текст селектора: " + elem.innerText);
</script>
</body>
```

Анотація статті

Перший абзац

Другий абзац

Текст селектора: Анотація статті

### 6.2.2 Властивості об’єктів document

Крім раніше розглянутих методів об'єкт *document* дозволяє звернутися до певних елементів веб-сторінки через властивості:

**documentElement** – надає доступ до кореневого елемента <html>;

**body** – надає доступ до елемента <body> на веб-сторінці;

**images** – містить колекцію всіх об'єктів зображень (елементів img);

**links** – містить колекцію посилань – елементів <a> та <area>, у яких визначено атрибут href;

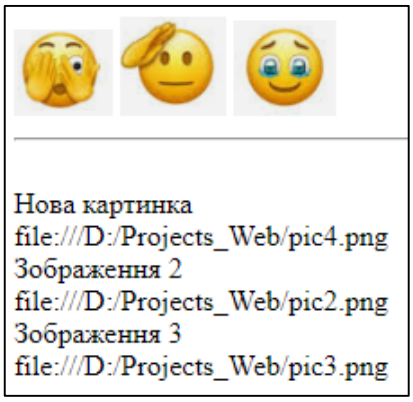
**anchors** – надає доступ до колекції елементів <a>, у яких визначено атрибут name;

**forms** – містить колекцію всіх форм на веб-сторінці.

Наприклад, отримаємо усі зображення на сторінці. Подібно до того, як у коді *html* ми можемо встановити атрибути у елемента *img*, так і в коді *javascript* ми можемо через властивості *src* та *alt* отримати та встановити значення цих атрибутів.

```
...
<body>
    <img src="pic1.png" alt="Зображення 1" />
    <img src="pic2.png" alt="Зображення 2" />
    <img src="pic3.png" alt="Зображення 3" /><hr>
```

```
<script>
    // Отримуємо масив усіх зображень
var images = document.images;
    // змінюємо перше зображення
images[0].src="pic4.png";
images[0].alt="Нова картинка";
    // перебираємо усі зображення
for(var i=0; i<images.length;i++){
    document.write("<br/>" + images[i].alt);
    document.write("<br/>" + images[i].src);
}
</script>
</body>
```




### 6.2.3 Властивості innerText та innerHTML

Для отримання або встановлення текстового вмісту елемента можна використовувати властивість *innerText*, а для отримання або встановлення коду *html* – властивість *innerHTML*. Наприклад,

```
<body>
<div class="article">
    <h3>Заголовок статті</h3>
    <p>Перший абзац</p>
    <p>Другий абзац</p><hr>
</div>
<script>
      // Отримуємо масив
var articleDiv = document.querySelector("div.article");
document.write(articleDiv.innerText+'<br>');
document.write("_____");
document.write(articleDiv.innerHTML);
</script>
</body>
```

### 6.2.4 Об’єкт Node

Кожен окремий вузол, чи то html-елемент, його атрибут чи текст, у структурі DOM подано об'єктом *Node*. Цей об'єкт надає ряд властивостей, за допомогою яких ми можемо отримати інформацію про сайт:

**childNodes** – містить колекцію дочірніх вузлів;
**firstChild** – повертає перший дочірній вузол поточного вузла;
**lastChild** – повертає останній дочірній вузол поточного вузла;
**previousSibling** – повертає попередній елемент, який знаходиться на одному рівні з поточним;
**nextSibling** – повертає наступний елемент, який знаходиться на одному рівні з поточним;
**ownerDocument** – повертає кореневий вузол документа;
**parentNode** – повертає елемент, який містить поточний вузол;

**nodeName** – повертає ім'я вузла;

**nodeType** – повертає тип вузла у вигляді числа (1 – елемент, 2 – атрибут, 3 – текст);

**nodeValue** – повертає або встановлює значення вузла у вигляді простого тексту.

Так, наприклад, використовуючи властивості *nextSibling* і *previousSibling* можна пройтися вузлами у прямому або зворотному порядку.

Крім того, об'єкт *document* має такі ***методи***:

**createElement(elementName)** – створює елемент html, тег якого передається як параметр. Повертає створений елемент;

**createTextNode(text)** – створює та повертає текстовий вузол. Як параметр передається текст вузла.

Наприклад,

```
var elem = document.createElement("div");
var elemText = document.createTextNode("Привіт світ");
```

У цьому фрагменті змінна *elem* зберігатиме посилання на елемент *div*. Проте, створення елементів недостатньо, їх ще потрібно додати на веб-сторінку. Елементи можна видаляти, замінювати. Для цього існують ще такі методи:

**appendChild(newNode)** – додає новий вузол newNode до кінця колекції дочірніх вузлів;

**insertBefore(newNode, referenceNode)** – додає новий вузол newNode перед вузлом referenceNode;

**removeChild()** – видаляє один із дочірніх вузлів;

**replaceChild(newNode, oldNode)** – цей метод як перший параметр приймає новий елемент *newNode*, який замінює старий елемент *oldNode*, що передається як другий параметр.

Але, як вже було сказано, під час роботи безпосередньо з вузлами DOM потрібно дуже ретельно дотримуватись ієрархії і слідкувати за успадкуванням елементів, що є нетривіальною задачею.

### 6.2.5 Об’єкт Element і керування елементами

Крім методів та властивостей об'єкта *Node* у JavaScript можна використовувати властивості та методи об'єктів *Element*. Важливо не плутати ці два об'єкти: *Node* та *Element*. *Node* подає всі вузли веб-сторінки, в той час як об'єкт *Element* подає лише html-елементи. Тобто об'єкти *Element* – це фактично ті самі вузли – об'єкти *Node*, у яких тип вузла (властивість *nodeType*) дорівнює 1.

Однією з ключових властивостей об'єкта *Element* є властивість *tagName*, яка повертає тег елемента.

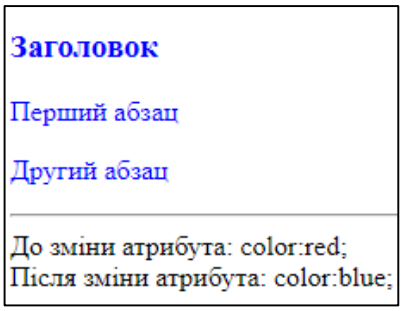
Серед методів об'єкта Element наявні методи управління атрибутами:

**`getAttribute(attr)`**: повертає значення атрибуту attr;

**`setAttribute(attr, value)`**: встановлює для атрибута attr значення value. Якщо атрибуту немає, він додається;

**`removeAttribute(attr)`**: видаляє атрибут attr та його значення.

Приклад.



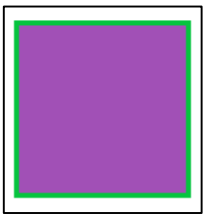

```
...
<body>
<div class="article" style="color:red;">
  <h3>Заголовок</h3>
  <p>Перший абзац</p>
  <p>Другий абзац</p><hr>
</div>
<script>
var articleDiv = document.querySelector("div.article");
    // отримуємо атрибут style
var styleValue = articleDiv.getAttribute("style");
document.write("До зміни атрибута: " + styleValue);
    // видаляємо атрибут
articleDiv.removeAttribute("style");
    // додаємо заново атрибут style
articleDiv.setAttribute("style", "color:blue;");
styleValue = articleDiv.getAttribute("style");
document.write("<br>Після зміни атрибута: " + styleValue);
</script>
</body>
```

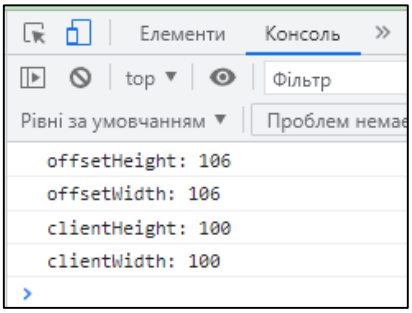
Елементи мають ряд властивостей, які визначають ***розмір елемента***. Але важливо розуміти різницю між усіма цими властивостями:

**`offsetWidth`** і **`offsetHeight`** – визначають відповідно ширину та висоту елемента в пікселях, водночас включається межа елемента;

**`clientWidth`** і **`clientHeight`** – також визначають ширину та висоту елемента в пікселях, але без урахування меж.

Приклад. Виведемо на сторінці прямокутник, а у консолі виведемо усі його розміри.






```
<html>
<head>
    <meta charset="utf-8" />
<style>
    #rect {
        width: 100px;
        height: 100px;
        background: #a150b6;
        border: 3px solid rgb(5, 196, 62);
    }
    </style>
</head>
<body>
<div id="rect"></div>
<script>
var rect = document.getElementById("rect");
console.log("offsetHeight: " + rect.offsetHeight);
```

```
console.log("offsetWidth: " + rect.offsetWidth);
console.log("clientHeight: " + rect.clientHeight);
console.log("clientWidth: " + rect.clientWidth);
</script>
</body>
</html>
```

Для визначення позиції елемента найбільш ефективним способом є метод **getBoundingClientRect()**. Цей метод повертає об'єкт із властивостями *top*, *bottom*, *left*, *right*, які вказують на зміщення елемента відносно лівого верхнього кута вікна браузера. Наприклад,

```
var rect = document.getElementById("rect");
var clientRect  = rect.getBoundingClientRect();
console.log("top: " + clientRect.top);
console.log("bottom: " + clientRect.bottom);
console.log("left: " + clientRect.left);
console.log("right: " + clientRect.right);
```

Для роботи з атрибутами вузлів використовують властивість ***style.***

Властивість *style* є складним об'єктом для керування стилем і безпосередньо зіставляється з атрибутом *style* html-елемента. Цей об'єкт містить набір властивостей CSS: *element.style.властивість CSS*.

Наприклад, встановимо колір шрифта:

```
var root = document.documentElement;
root.style.color = "blue";          // встановлюємо стиль
document.write(root.style.color);   // отримуємо значення стиля
```

Ряд властивостей css у назвах мають дефіс, наприклад *font-family*. JavaScript для цих властивостей дефіс не використовує. Тільки перша літера, яка йде після дефісу, переводиться у верхній регістр. Наприклад,

```
var root = document.documentElement;
root.style.fontFamily = "Verdana";
```

### 6.3 Події і їх обробка

Для взаємодії з користувачем JavaScript визначено механізм подій. Наприклад, коли користувач натискає кнопку, виникає подія натискання кнопки. Тут кнопка – *джерело події*, натискання на кнопку – *подія*. Розробник має визначити у коді, що необхідно зробити, щоб відбулась якась реакція на подію, тобто підготувати *блок прослуховування події* або блок реакції на подію (рис. 6.3).


Рисунок 6.3 – Модель делегування подій

Отже, у коді JavaScript ми можемо визначити виникнення події та якось її обробити.

JavaScript має такі типи подій:

- події миші (переміщення курсору, натискання миші тощо);
- події клавіатури (натискання або відпускання клавіші клавіатури);
- події життєвого циклу елементів (наприклад, подія завантаження веб-сторінки);
- події елементів форм (натискання кнопки на формі, вибір елемента у списку, що випадає, і т. д.);
- події, що виникають під час зміни елементів DOM;
- події, що виникають під час торкання сенсорних екранів;
- події, що виникають у разі виникнення помилок.

### 6.3.1 Основні типи подій

Події миші над елементом:

| | |
|---|---|
| **onclick** | клік миші по елементу |
| **ondblclick** | подвійний клік миші по елементу |
| **onmousedown** | натиснуто кнопку миші на елементі |
| **onmouseup** | відпущено кнопку миші |
| **onmousemove** | переміщення курсору миші |
| **onmouseover** | наведено курсор миші на елемент |
| **onmouseout** | відведено курсор миші від елементу |
| **oncontextmenu** | визвано контексне меню |

Події клавіатури:

| | |
|---|---|
| **onkeydown** | натискається клавіша на клавіатурі |
| **onkeypress** | натиснута клавіша на клавіатурі |
| **onkeyup** | відпущена клавіша на клавіатурі |

Події форми:

| | |
|---|---|
| **onsubmit** | надсилання даних форми |
| **onreset** | очищення даних форми |
| **oninvalid** | елемент не пройшов валідацію |

Події буфера обміну:

| | |
|---|---|
| **onbeforecopy** | дані будуть копіюватася у буфер обміну |
| **oncopy** | копіюється текст елемента у буфер обміну |
| **oncut** | вирізається текст у буфер обміну |
| **onbeforecut** | дані будуть вирізатися у буфер обміну |
| **onbeforepaste** | дані будуть вставлені з буферу обміну |
| **onpaste** | дані вставляються з буферу обміну у елемент |

Інші події елементів:

| | |
|---|---|
| **ontoggle** | відкрито / закрито елемент |
| **onscroll** | прокрутка |
| **onselect** | виділення тексту |
| **onchange** | значення було змінено у елементі |
| **oninput** | введення даних |
| **onfocus** | елемент отримав фокус |
| **onblur** | елемент втратив фокус |

Існує ще ціла низка обробників інших подій, як то: події сенсорного екрана (тачпаду) над елементом, події тегів *audio* і *video*. Детальну інформацію про використання інших обробників можна знайти на офіційних сайтах та у інших джерелах.

Крім того, для обробки подій використовується об'єкт *Event*. Під час обробки події браузер автоматично передає у функцію обробника як параметр саме об'єкт Event, який інкапсулює всю інформацію про подію. І за допомогою його властивостей можна отримати додаткову інформацію.

### 6.3.2 Обробка подій

Є два способи обробки подій:

*1-й спосіб*.

Він полягає у тому, що можна задати в коді HTML атрибут елемента керування, який складається з префікса *on* і назви події.

Приклади.

1. Підраховуємо і виводимо на сторінку кількість кліків мишею на ній:

```
<script type="text/javascript">
  var clickCount = 0;
  function documentClick(){
    document.getElementById('clicked').value = ++clickCount;
  }
  document.onclick = documentClick();
</script>
  Ви клацнули на цій сторінці
  <input id="clicked" size="3" value="0"> разів
```

2. За наведення мишею на посилання отримуємо підказку:

```
<script type="text/javascript">
  function doMouse(){
      alert("Натисніть тут для переходу на Google");
}
</script>
<a href=http://www.google.com onMouseOver = "doMouse()">
   Перехід на Google.com </a>
```

*2-й спосіб*.

Обробити подію можна в тегах JavaScript, тобто вказати як властивість певного елемента документа.

Розглянемо декілька прикладів обробки подій.

Приклад 1. Підказки під час наведення на гіперпосилання

```
<a href = http://www.google.com
   onMouseOver = "alert('Натисніть тут для переходу на google')">
       Перехід на сторінку rambler.ru </a>
<a href = http://www.vntu.edu.ua
   onMouseOver = "alert('Натисніть тут для переходу на сайт ВНТУ')">
        Перехід на сторінку ВНТУ </a>
```

Приклад 2. Повідомлення у вікні статусу під час завантаження сторінки

```
<body onLoad=" alert('Завантажується  нове вікно')"
      onMouseOut = "window.status='********'"
</body>
```

Приклад 3. Підказки під час перевірки введення паролю

```
Ваш пароль:
<input type="password" onBlur = "alert('Ви не ввели пароль!')">
Новий пароль:
<input type="text" onChange = "alert('Перевірка правильності …')">
```

Приклад 4. Анімація кнопки за допомогою обробки подій.

Нехай є три зображення кнопки – в звичайному стані, коли на кнопку наводиться миша, коли кнопка натиснута:


but0.gif


but1.gif


but2.gif

Анімація кнопки опишеться таким чином:

```
<img src="Img/but1.gif" width=92 height=52 border=0
   onMouseDown="src='Img/but0.gif' "
   onMouseOver="src='Img/but2.gif' "
   onMouseOut="src='Img/but1.gif'"
   onMouseUp="src='Img/but1.gif'"
   onclick="..." />
```

Приклад 5. Управління зображеннями

В цьому прикладі під час вибору гіпертекстового посилання (посилання знаходяться в одному місці сторінки (у нашому прикладі – у лівому блоці) відбувається виклик функції *LoadImg*(), яка змінює значення атрибута *src* тегу <*img*>, причому зображення знаходиться у якомусь іншому місці сторінки (у нашому прикладі – у правому блоці).

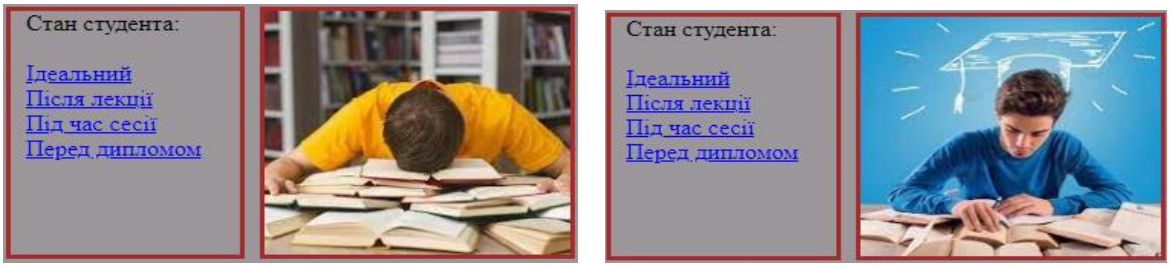
Фрагмент коду html:

```
<div style="display: flex; flex-flow: row wrap;">
    <div style="width: 250px; height: 175px;
           border:3px solid brown; margin: 10px 10px 10px 10px">
     <p>Стан студента:<br><br>
        <a href='javascript: LoadImage("Img/Stud1.jpg")'>
                        Ідеальний</a><br>
        <a href='javascript: LoadImage("Img/Stud2.jpg")'>
                        Після лекції</a><br>
        <a href='javascript: LoadImage("Img/Stud3.jpg")'>
                        Під час сесії</a><br>
       <a href='javascript: LoadImage("Img/Stud4.jpg")'>
                        Перед дипломом</a></p>
    </div>
    <div style="width: 200px; height: 175px;
           border:3px solid brown; margin: 10px 10px 10px 10px">
        <img src="Img/Stud1.jpg" width="200px" height="175px">
    </div>
 </div>
```

У функції *LoadImage()* застосовується об'єктна модель документа: створюється контейнер (масив) тегів зображень на сторінці (а у нас лише одне зображення), і у елемент з індексом 0 виводиться зображення з вказаним іменем:

```
<script type="text/javascript">
    function LoadImage(a) {
        document.images[0].src=a;
    }
</script>
```

Як результат – під час вибору певного посилання на одному і тому самому місці з'являються різні зображення:




Приклад 6. Створення найпростішого слайд-шоу

Для показу змінюваних картинок нам потрібно звертатися до тегу <*img*>. Щоб звертатися до конкретного тегу, краще його «обізвати», наприклад, *imageShow*. Повністю тег виглядає так:

```
<img src="pic1.png" width="150" height="150px" id="imageShow"><br>
```

Програма буде «підставляти» почергово посилання на картинки. Щоб запускати програму потрібна кнопка, яка буде викликати функцію, і ще кнопка, щоб зупиняти функцію.

```
<input type="button" value="GO"
       onClick="timer=setInterval('showMustGo()',1000)">
<input type="button" value="STOP"
       onClick="clearInterval(timer)">
```

Для запуску ми використовуємо подію *onclick* з інструкцією

```
timer = setInterval('showMustGo()', 2000)
```

Тут використано вбудовану функцію *setInterval()*, аргументами якої є функція та інтервал (в мілісекундах!), через яку до цієї функції ми будемо звертатися. Щоб зупинити «таймер», ми присвоюємо значення *timer* =, а для зупинки використовуємо `clearInterval(timer).`

Самі посилання ми будемо зберігати в масиві *imagesMassiv.*

Звернення до елементів масиву: *imagesMassiv[n]*, де *n* – порядковий номер елемента (масиви нумеруються з 0!).

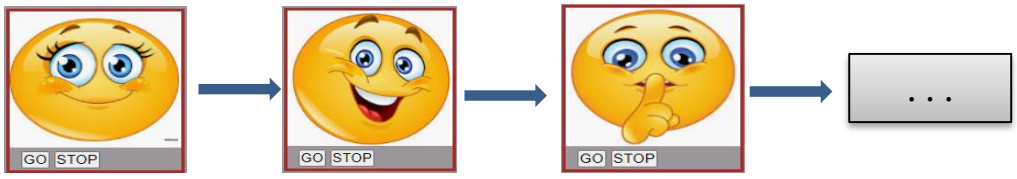
Перейдемо до функції *showMustGo():*

- перш ніж підставити чергове посилання, ми маємо переконатися, що не вийшли за рамки масиву;
- якщо вийшли за рамки масиву, то поставити лічильник на перший елемент;
- після показу картинки збільшити значення лічильника на 1.

Отже, код скрипта такий:

```
<script type="text/javascript">
   var imagesMassiv = new Array('pic1.png', 'pic2.png','pic3.png',
              'pic4.png','pic5.jpg','pic6.jpg','pic7.jpg');
   var countImages = 0;
   var timer;
   function showMustGo() {
      if (countImages == imagesMassiv.length) {
                countImages = 0;
      }
      document.images['imageShow'].src = imagesMassiv[countImages];
            countImages = countImages + 1;
    }
</script>
```

Перевіряємо роботу нашого слайдера. Бачимо, що зображення починають прокручуватись після натискання кнопки GO і зупиняється під час натискання кнопки STOP.




Приклад 7. Зміна стилю елемента

Нехай нам потрібно змінювати стиль абзаца залежно від якихось умов, наприклад, змінюватимемо колір тексту в абзаці. Передбачимо на сторінці декілька кнопок, які відповідатимуть за зміну кольору тексту абзаца. Введемо в код сторінки такий фрагмент:

```
<p id="text"><br><b>Цей текст змінюватиме колір</b><br><br></p>
    <input type="button" value="Red" style='background-color:red;'
            onClick="changeColor(1)">
    <input type="button" value="Green" style='background-color:green;'
```

```
        onClick="changeColor(2)">
    <input type="button" value="Blue" style='background-color: blue;'
        onClick="changeColor(3)">
```

А в код скрипта введемо нову функцію:

```
function changeColor(num){
  switch(num){
  case 1: document.getElementById('text').style.color="red"; break;
  case 2: document.getElementById('text').style.color="green"; break;
  case 3: document.getElementById('text').style.color="blue"; break;
  }
}
```

Маємо:


### 6.4 HTML-форми

Форма HTML являє собою документ, створений з використанням елементів HTML. Призначенням форми є збір інформації від користувачів. Після того як користувач заповнить форму і запускає процес її обробки, інформація з неї потрапляє в програму, що працює на сервері. Інша програма під назвою Common Gateway Interface (CGI) обробляє її. Таким чином користувач може інтерактивно взаємодіяти з сервером Web через Internet. Форми так само зручні і для розробників сайту під час розробки CMS (система управління вмістом – Content Management System), яка дозволяє підтримувати головну властивість сайту – актуальність.

Форми використовуються для:

- отримання відгуку користувача на надану інформацію;
- збору даних про користувача на сервері;
- інтерактивної взаємодії користувача з Web-сервером.

Для набуття інтерактивності на формах розташовують елементи керування.

*Кнопки*:

- кнопки відправлення (даних форми серверу) (SUBMIT);
- кнопки скидання значень (повернення початкових значень) (RESET);
- інші кнопки, для них не вказано дію, що виконують як встановлено (BUTTON и INPUT)

*Перемикачі* (INPUT):

- залежні («вкл/викл») (RADIO)
- незалежні, що можуть приймати і змінювати своє значення незалежно від інших перемикачів (CHECKBOX)

*Списки* і *Меню*, які надають користувачу перелік можливих варіантів вибору:

- SELECT, OPTGROUP і OPTION.

*Текстові поля* :

- однорядкові (за допомогою елементів INPUT);
- багаторядкові (TEXTAREA)
- інші (PASSWORD)

*Інші елементи керування*

Для створенні форм використовують теги <*form*> і </*form*>:

```
<form   method=post  action="handler.php">
    . . .  елементи керування . . .
</form>
```

*method* – вказує браузеру, який вид HTTP запиту необхідно використовувати для відправлення форми – GET або POST. Головна відмінність методів POST і GET полягає у способі передачі інформації.
*GET* – параметри передаються через адресний рядок, тобто і в HTTP-заголовку (у вигляді URL?змінна=значення&змінна=значення&… );
POST – параметри передаються через тіло HTTP-запиту і ніяк не відображаються на вигляді рядка.

*action* – URL-адреса програми (URL сценарію), що оброблятиме інформацію з форми.

## 6.5 Елементи керування

### 6.5.1 Текстове поле однорядкове

**<INPUT [TYPE = TEXT]** … … … **>**

Цей елемент є полем для введення інформації користувачем.
Інші атрибути тега:

**name** – для визначення найменування змінної поля;

**maxlength** – обмежує число символів у полі (за замовчуванням – не обмежено);

**size** – визначає розмір видимої на екрані області (за замовчуванням визначається типом браузера);

**value** – початкове значення.

Цей елемент підтримує низку подій, зокрема:

**focus** – відбувається під час отримання фокусу;

**blur** – відбувається у разі втрати фокусу;

**change** – відбувається за зміни значення поля;

**select** – відбувається під час виділення тексту у текстовому полі;

**keydown** – відбувається під час натискання клавіші клавіатури;

**keypress** – відбувається під час натискання клавіші клавіатури для друкованих символів;

**keyup** – відбувається під час відпускання раніше натиснутої клавіші клавіатури.

Приклад.

```
Ваше ім'я <INPUT  NAME=Name SIZE=35>
```

Результат:


### 6.5.2 Поле введення паролю

**<INPUT TYPE = PASSWORD ... ... ... >**

Цей елемент використовується для організації введення пароля без виведення на екран складових його символів (замість символів виводяться зірочки або крапочки).

Інші атрибути:

**name** – для визначення найменування змінної поля;

**maxlength** – обмежує число символів у полі;

**size** – розмір видимої на екрані області.

Приклад.

```
Підпис <INPUT NAME=login>
Пароль <INPUT TYPE=password NAME="Слово">
```

Результат:


### 6.5.3 Багаторядкове текстове поле

**<TEXTAREA NAME=... COLS=... ROWS=...>**

**</TEXTAREA>**

Атрибути:

**name** – найменування поля;

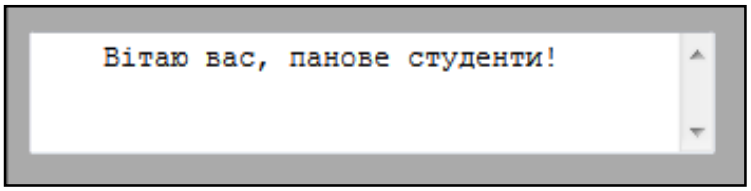
**cols** – число колонок (число символів);

**rows** – кількість видимих рядків текстової області.

Приклад.

```
<TEXTAREA NAME="тема" COLS="38" ROWS="3">
</TEXTAREA>
```

Результат:



### 6.5.4 Незалежні прапорці

**<INPUT TYPE = CHECKBOX … … … >**

Прапорці *checkbox* пропонують користувачеві ряд варіантів, і дозволяє вибір декількох із них.

Інші атрибути:

**name** – для визначення найменування змінної поля;

**value** – початкове значення;

**checked** – ініціалізація прапорця.

Натискання на прапорець генерує подію *click*.

Приклад.

```
<p>Ukraina <INPUT NAME='Ukr' TYPE=checkbox VALUE="Україна">
Russia <INPUT NAME='Rus' TYPE=checkbox VALUE="Росія">
Країни СНД <INPUT NAME="cnd" TYPE=checkbox VALUE="СНД">
</p>
```

Результат:


### 6.5.5 Залежні прапорці (селектор)

**<INPUT TYPE = RADIO … … … >**

Цей атрибут використовується для організації вибору одного єдиного варіанта із декількох можливих. Інші атрибути:

**name** – для визначення найменування змінної поля;

**value** – початкове значення;

**checked** – ініціалізація прапорця.

Натискання на селектор генерує подію *click*.

Приклад.

```
Стать чоловіча <INPUT NAME="state" TYPE=radio
                        VALUE="Чоловіча">
Стать жіноча <INPUT NAME="state" TYPE=radio
                         VALUE="Жіноча" >
```

Результат:


### 6.5.6 Список

```
<SELECT ... >
   <OPTION ...>
   <OPTION ...>
   ...
<SELECT >
```

Для організації списків з прокруткою і низхідним меню можна використовувати елемент <*SELECT*>. Для визначення списку пунктів використовуються елементи <*OPTION*>. Разом з атрибутом SELECT можна використовувати такі атрибути:

**multiple** – дозволяє мультивибір;

**name** – найменування об'єкта.

**Size** – число видимих користувачу пунктів списку (якщо *size = 1* – список у вигляді низхідного меню, за *size > 1* – звичайний список);

**<option>** – всередині тега <*select*>, може мати додаткові атрибути:

**selected** – первинний вибір;

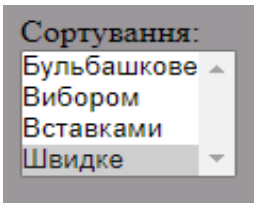
**value** – значення, що повертається.

Приклад.

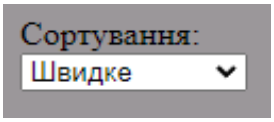
```
<p> Сортування:<br>
<SELECT NAME="Choice" size=4>
    <OPTION> Бульбашкове
    <OPTION> Вибором
    <OPTION VALUE="Insert"> Вставками
    <OPTION SELECTED> Швидке
</SELECT></p><br>
```

Результат:

для *size=4*




для *size=1*




Тобто, за *size=1* звичайний список перетворився у комбінований список.

Елемент *select* підтримує три події: *blur* (втрата фокуса), *focus* (отримання фокуса) та *change* (зміна виділеного елемента у списку).

### 6.5.7 Зображення

**<INPUT TYPE = IMAGE** … … … **>**

Залежно від вмісту форми може статися так, що користувачеві буде потрібно клацнути мишею на зображенні, щоб завершити роботу з формою. Для організації цього використовується елемент *IMAGE*. Після клацання користувача по зображенню браузер зберігає координати потрібної точки екрана і приймає всю форму. Разом з елементом *IMAGE* використовуються такі атрибути:

**name** – для визначення найменування змінної поля;

**src** – url файлу – джерела зображення;

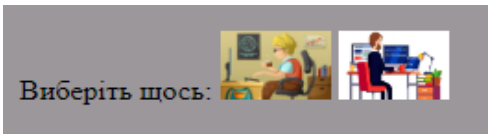
**align**, **width**, **height** – параметри зображення.

Приклад.

```
Виберіть щось:
<INPUT TYPE=IMAGE NAME=pt1 SRC='pr1.jpg' height="40px" width="60px">
```

```
<INPUT TYPE=IMAGE NAME=pt2 SRC='pr2.jpeg' height="40px" width="60px">
```

Результат:




### 6.5.8 Вибір файлу

**<INPUT TYPE = FILE … … … >**

Інші атрибути мають такі самі значення, як і для інших елементів: **name, value, maxlength, size.**

Приклад.

```
Виберіть файл
<INPUT TYPE=file NAME=Filename MAXLENGTH=50 SIZE=35>
```

Результат:


### 6.5.9 Веб-адреса

**<INPUT TYPE = URL … … … >**

Інші атрибути (збігаються з TEXT, але може здійснюватись валідація введених значень): **name, value, maxlength, size.**

Приклад.

```
<p>Виберіть сайт:
    <input type="url" name="site" required>
</p>
```

### 6.5.10 Кнопка відновлення

**<INPUT TYPE = RESET … … … >**

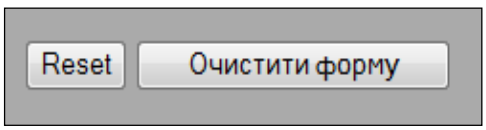
Інші атрибути:

**value** – надпис на кнопці.

Приклад.

```
<INPUT TYPE=reset >
<INPUT TYPE=reset VALUE="Очистити форму">
```

Результат:




### 6.5.11 Кнопка завершення

```
<INPUT  TYPE = SUBMIT …     …    …    >
```

Цей елемент використовується під час закінчення введення користувачем даних. Браузер виводить цей елемент як кнопку, на якій користувач може клацнути, щоб завершити редагування. Додаткові атрибути:

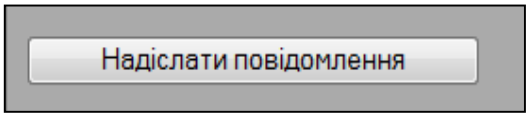
**name** – назва змінної поля;

**value** – надпис на кнопці.

Приклад.

```
<INPUT TYPE = submit VALUE = "Надіслати повідомлення">
```

Результат:




### 6.5.12 Кнопка звичайна

Крім кнопок відновлення *(type='reset')* і завершення *(type='submit')*, на web-сторінках і у формах можна використовувати звичайні кнопки. Їх можна створити декількома способами:

а) за допомогою елемента <*input*>:

```
<INPUT TYPE="BUTTON" VALUE="Текст на кнопці">
```

Атрибути кнопки `<input>`:

**name** – ім'я кнопки, призначене для того, щоб обробник форми міг її ідентифікувати;

**disabled** – блокує кнопку і не дозволяє на неї натискати;

**form** – ідентифікатор форми для зв'язування кнопки з елементом *<form>;*

**type** – для звичайної кнопки значенням є *button*;

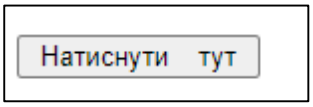
**value** – напис на кнопці;

**autofocus** – кнопка отримує фокус після завантаження документа.

Наприклад, внаслідок виконання інструкції

```
<p>
   <input type="button" value=" Натиснути  тут ">
</p>
```

з'явиться кнопка:

б) за 

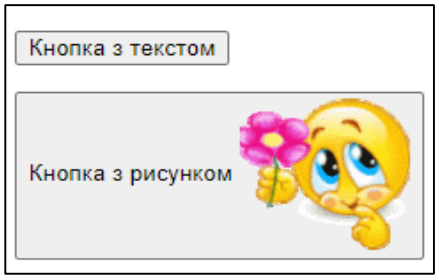
допомогою елемента <*button*>:

```
<BUTTON>  Напис  на кнопці </BUTTON>
```

Атрибути, перераховані для кнопки <*input*>, діють і у цьому випадку, але атрибут *value* визначає тільки значення, яке відправляється на сервер, а не напис на кнопці. Якщо потрібно вивести на кнопці зображення, то <*img*> додається всередину <*button*>.

Наприклад, фрагмент коду дасть наведений результат

```
<p>
    <button>Кнопка з текстом</button>
</p>
<p>
    <button> Кнопка з рисунком
     <img src="images/but.png"
          style="vertical-align:middle">
   </button>
</p>
```




### 6.5.13 Приховані поля

```
<INPUT  TYPE = HIDDEN…      …
…    >
```

Додаткові атрибути:

**name** – назва змінної поля;

**value** – значення.

Цей елемент корисний, наприклад, за наявності декількох форм для подальшої обробки. Користувач змінити його не може.

### 6.5.14 Групування елементів

```
<FIELDSET>
     <LEGEND> Надпис </LEGEND>
      ...
</FIELDSET>
```

HTML тег <*fieldset*> використовується для групування пов'язаних елементів в формі. Таке групування полегшує роботу з формами, що містять велику кількість даних, наприклад, один блок може бути призначений для введення текстової інформації, а інший – для прапорців.

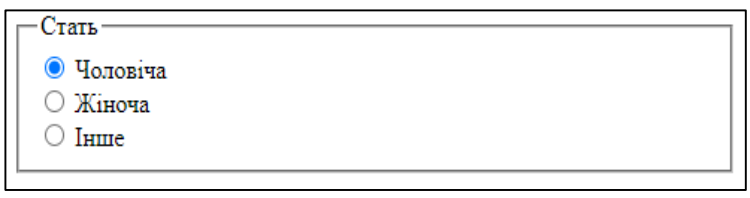
Тег <*fieldset*> рисує рамку навколо пов'язаних елементів. Її вигляд залежить від операційної системи, а також браузера.

Кожному з елементів <*fieldset*> і <*legend*> можна надати певного стилю: розміру, кольору, товщини тощо.

Приклад.

```
<fieldset>
   <legend>Стать</legend>
   <input type='radio' name='gender" value='male' checked> Чоловіча<br>
   <input type='radio' name='gender' value='female'> Жіноча<br>
   <input type='radio' name='gender' value='other'> Інше
 </fieldset>
```

Результат:




### 6.5.15 Інші типи полів

Починаючи зі стандарту HTML5, з'явилися деякі нові зручні елементи керування, використовувані на формах. Всі вони використовують тег <INPUT> і мають певний тип:

**type="email"** – введення E-Mail – текстове поле, на клавіатурі мобільних пристроїв з'являється символ @;

**type="tel"** – номер телефону. На мобільних пристроях відкривається клавіатура з числами;

**type="number"** – числове поле (крім клавіатури з цифрами з'являється можливість перемикати значення поля (атрибути:

*min* і *max* – нижнє і верхнє можливе значення, *step* – крок зміни, *value* – початкове значення;

**type="range"** – числовий повзунок (атрибути: *min*, *max*, *step* і *value*);

**type="search"** – поле пошуку. Google Chrome додає хрестик для очищення введеного рядка. На мобільних пристроях з'являється кнопка пошуку.

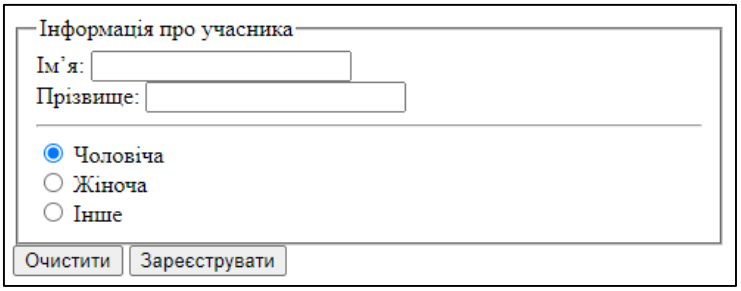
## 6.6 Реалізація і обробка форм

### 6.6.1 Форма у таблиці

Під час створення форми на web-сторінці зручно використовувати таблиці. У цьому випадку елементи керування можна вирівнювати однаково і надавати їм певний розмір. Наприклад,

```
<form> <fieldset>
       <legend> Інформація про учасника </legend>
       Ім'я:  <input type='text' name='firstname'><br>
       Прізвище: <input type='text' name='lastname'>
       <hr>
       <input type='radio' name='gender" checked> Чоловіча<br>
       <input type='radio' name='gender'> Жіноча<br>
       <input type='radio' name='gender' > Інше
       </fieldset>
    <input type="reset" value="Очистити">
    <input type="submit" value="Зареєструвати">
  </form>
```

У цьому випадку форма набуде такого вигляду:

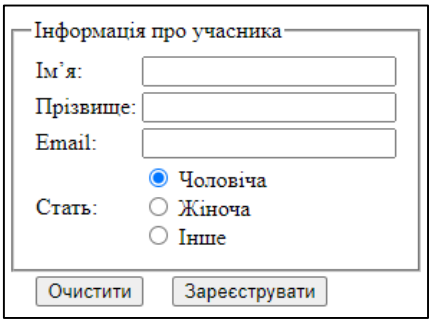



А тепер використаємо таблицю:

```
<form>
   <fieldset>
      <legend>Інформація про учасника</legend>
```

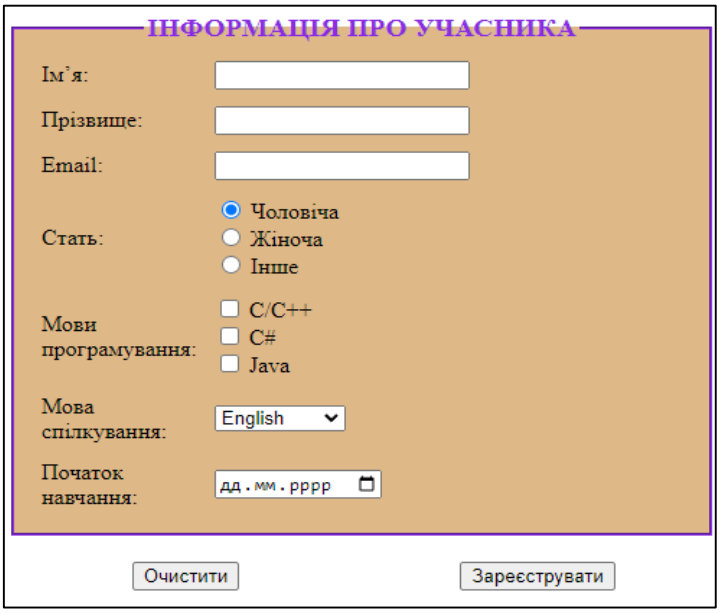
```
<table><tr><td>Ім'я: </td>
     <td><input type='text' name='firstname' required></td>
   </tr>
   <tr><td>Прізвище: </td>
       <td><input type='text' name='lastname' required></td>
   </tr>
   <tr><td>Email: </td>
       <td><input type='Email:' name='male'></td>
   </tr>
   <tr><td>Стать:</td>
     <td><input type='radio' name='gender" checked> Чоловіча<br>
     <input type='radio' name='gender' > Жіноча<br>
     <input type='radio' name='gender' > Інше </td>
</tr></table></fieldset>
<table><tr><td width="50%">input type="reset" value="Очистити"><td>
    <td><input type="submit" value="Зареєструвати"></td>
</tr></table>
</form>
```

Тепер форма виглядатиме дещо охайніше:




### 6.6.2 Обробка елементів форми на прикладах

Покажемо обробку елементів форми на прикладах.
Нехай, наприклад, у нас є форма такого вигляду.




Фрагмент коду, що реалізує таку форму, може бути таким:

```
<form action="" metod="" >
    <fieldset style="background-color: burlywood;">
      <legend style="text-align: center;">
           <b>ІНФОРМАЦІЯ ПРО УЧАСНИКА</b>
       </legend>
       <table cellpadding="5px">
           <tr><td>Ім’я: </td>
             <td><input type='text' id='firstname' required></td>
           </tr>
           <tr><td>Прізвище: </td>
            <td><input type='text' name='lastname' required></td>
          </tr>
          <tr><td>Email: </td>
             <td><input type='email' ></td>
          </tr>
          <tr>
           <td>Стать:</td>
           <td><input type='radio' name='gender' value='чол.' checked> Чоловік<br>
                  <input type='radio' name='gender' value='жін.'> Жінка<br>
                  <input type='radio' name='gender' value='ін.'> Інше
            </td>
          </tr>
          <tr>
            <td>Мови<br>програмування:</td>
            <td><input type="checkbox" name="lang" value="C/C++">C/C++<br>
                  <input type="checkbox" name="lang" value="C#">C#<br>
                  <input type="checkbox" name="lang" value="Java">Java
            </td>
          </tr>
          <tr>
              <td>Мова<br>спілкування:</td>
              <td><select id="talk">
                     <option value="Англійська">English </option>
                     <option value="Німецька">Deutsch </option>
                     <option value="Італійська">Italian</option>
                     <option value="Українська">Українська </option>
                  </select>
              </td>
          </tr>
          <tr>
               <td>Початок<br>навчання:</td>
               <td><input type='date' id="data"></td>
          </tr>
       </table>
    </fieldset>
    <p>
     <table width="100%">
     <tr>
       <td width="50%">
              <center><input type="reset" value="Очистити"></center>
        </td>
       <td>
              <center> <input type="submit" value="Зареєструвати"
                        onclick="return getInfo()"></center>
        </td>
     </tr>
     </table>
     </p>
     </form>
```

Для виведення результату заповнення форми підготуємо окремий абзац, присвоївши йому ідентифікатор:

```
<h2 style="text-align: center;">РЕЗУЛЬТАТИ РЕЄСТРАЦІЇ</h2>
<hr/>
<p id="rezult"></p>
```

У цьому прикладі створено таблицю на один рядок, у першому стовпці якого розташовується форма, а у другому стовпці виводиться результат заповнення реєстраційної форми.

Код для обробки форми можна написати в окремому css-файлі або в тегах *<script> … </script>*.

Для зручності і для кращого розуміння коду ведемо в код JavaScript для кожного з текстових полів окрему змінну, а потім, об'єднавши ці змінні в один рядок, отримаємо у вигляді текстового буфера всю інформацію, яку потім виведемо у деякий елемент на документі (наприклад, виведемо це все у тег * … *).

### *Обробка текстових полів*

На наведеній формі присутні три текстових поля: *Ім'я, Прізвище* та *Email*. Для того, щоб показати різні способи отримання інформації з текстових полів, використаємо доступ до них за ідентифікатором (**`getElementById(value)`**) та за іменем (**`getElementsByName(name)`**) (хоча можна і за селектором, і за класом, і за допомогою послідовного доступу).

Варто звернути увагу на те, що, оскільки у текстових полях для введення імені і прізвища стоїть атрибут *required*, то у випадку, якщо ці поля не заповнені, браузер видасть повідомлення про це. Так само браузер відслідкує неправильність заповнення поля типу *email* (рис. 6.3).

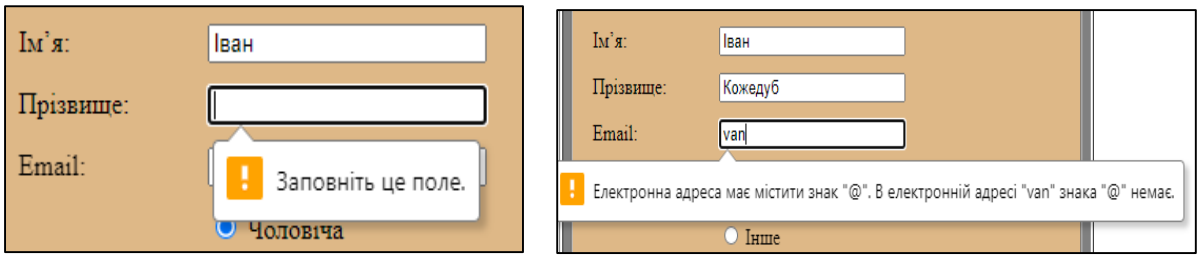



Рисунок 6.3 – Повідомлення браузера про помилки у формі

Для отримання інформації з наведених на формі текстових полів можна використати такий фрагмент коду:

```
    // вибір елемента за ідентифікатором
let firstName=document.getElementById('firstname').value;
    // вибір елемента за іменем
let lastName=document.getElementsByName('lastname')[0].value;
    // вибір за селектором
let eMail=document.querySelectorAll('input[type="email"]')[0].value;
```

### *Обробка перемикачів і прапорців*

В процесі обробки перемикачів з незалежною *(type='checkbox')* і залежною (*type='radio'*) фіксацією досить зручно дотримуватись такого алгоритму:

– отримати масив елементів-перемикачів;
– організувати цикл по елементах масиву;
– перевіряючи значення атрибуту *checked* кожного з елементів масиву, отримати відповідні значення атрибуту *value*.

На наведеній формі група залежних перемикачів дозволяє ввести стать (*name='gender'*), а незалежні прапорці (*name='lang'*) дозволяють обрати одну або декілька мов програмування для вивчення. Отримання цієї інформації з форми може бути здійснено таким чином:

```
let gender='';
let array_sex=document.getElementsByName('gender');
for (var i=0; i<array_sex.length; i++){
    if (array_sex[i].checked){
          gender=array_sex[i].value;
    }
}
```

Отримаємо інформацію про вибрані мови програмування:

```
let lang='';
var arr_lang=document.getElementsByName('lang');
for (i=0; i<arr_lang.length; i++){
    if (arr_lang[i].checked){
        lang+=arr_lang[i].value+'; ';
    }
}
```

***Обробка інших елементів форм***

Обробка списків та інших елементів керування у цьому прикладі виконана найпростішим чином:

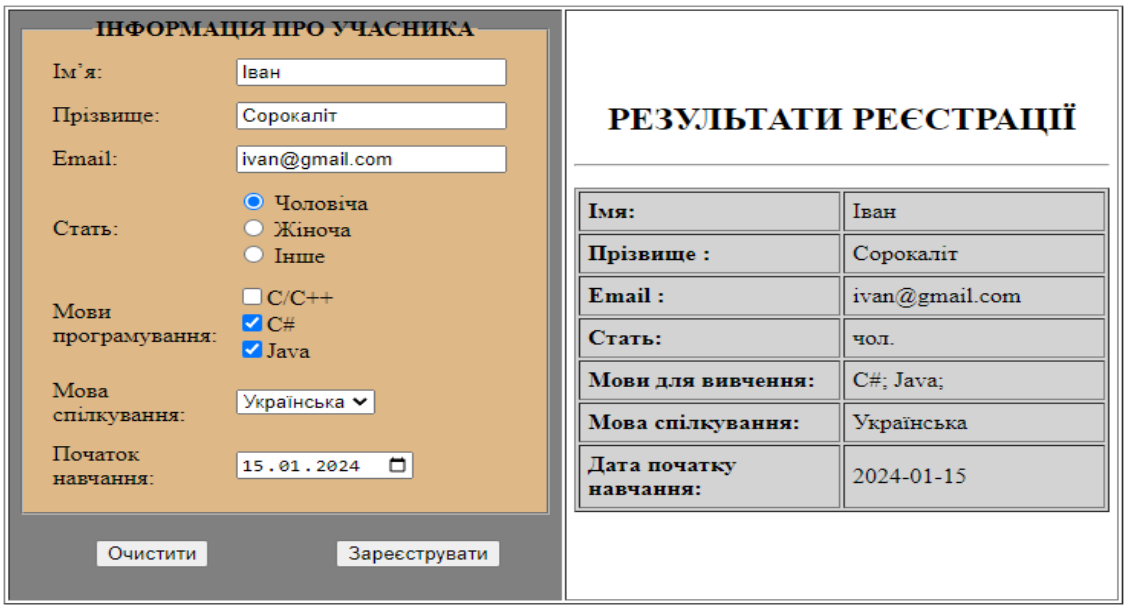
```
var talk=document.getElementById('talk').value;
var begin = document.getElementById('data').value;
```

Звичайно, якщо необхідно отримати більш детальні відомості про атрибути певних елементів керування, доведеться використовувати додаткові методи та властивості об'єктів, скориставшись додатковими інформаційними джерелами.

Отже, наразі маємо у змінних текстові значення атрибутів елементів керування у формі. Тепер залишається лише оформити їх і вивести як атрибут *innerHTML* тега <*p*> у відповідному місці web-сторінки:

```
document.getElementById("rezult").innerHTML=
   "<table border='1' width='100%' "
   + "cellpadding='5px' bgcolor='lightgray'>"
   + "<tr><td><b>Імя:</b> </td><td>" + firstName+"</tr>"
   + "<tr><td><b>Прізвище :</b></td><td>" + lastName+"</tr>"
   + "<tr><td><b>Email  :</b></td><td>" + eMail+"</tr>"
   + "<tr><td><b>Стать: </b></td><td>" + gender+"</tr>"
   + "<tr><td><b>Мови для вивчення: </b></td><td>" + lang+"</tr>"
   + "<tr><td><b>Мова спілкування: </b></td><td>" + talk+"</tr>"
   + "<tr><td><b>Дата початку навчання:</b></td><td>"+ begin+"</tr>"
   + "</table>";
```

Як результат маємо такий вигляд:




## Контрольні запитання

1. Що собою являє об'єктна модель документа?
2. Який об'єкт є в корені DOM?
3. Які види вузлів DOM існують? Продемонструвати це на прикладі власної html-сторінки.
4. В чому полягає послідовне звернення до елементів документа?
5. В чому полягає пряме звернення до елементів документа?
6. Які властивості має об'єкт *document*?
7. За що відповідає об'єкт *Node*?
8. За що відповідає об'єкт *Element*?
9. Що являє собою модель делегування подій? Які її складові?
10. Які способи обробки подій існують? Наведіть приклади.
11. Що таке html-форма? Яке її призначення?
12. Як створити форму? Що означають атрибути тегу <*form*>?
13. Які елементи форм існують?
14. Що собою являє тег <*INPUT*>?
15. Яким чином у форму вставити текстові поля, поля пароля, електронної пошти, дати?
16. Як на формі можна реалізувати списки (прості та комбіновані)?
17. Як реалізувати на формі перемикачі з незалежною і залежною фіксацією?
18. Наведіть приклади обробки елементів керування, використовуючи різні способи доступу: за ідентифікатором, за іменем, за селектором.
19. Чим відрізняється пошук елемента за іменем і за ідентифікатором?

## Практична робота № 6

### *Мета роботи*

– Ознайомитись з об’єктною моделлю документа JavaScript, дослідити способи доступу до елементів документа на web-сторінці.
– Вивчити основні теги для створення елементів керування у формах html-документів, розглянути приклади написання кодів для різних елементів, навчитися створювати комбіновані форми і обробляти їх.

### *Порядок виконання роботи*

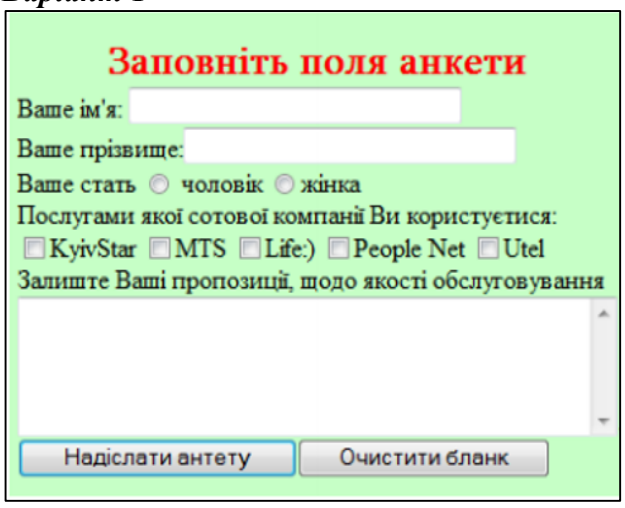
*Задача 1.* Відкоригувати коди для реалізації задач №1–2 лабораторної роботи № 5 таким чином, щоб вхідні дані вводились не через вікна повідомлень, а за допомогою відповідних елементів керування.

*Задача 2.* Підготувати web-сторінку, на якій здійснити розробку, реалізацію і обробку форми відповідно до індивідуального завдання.
Можна взяти до виконання власно придуману форму, яка, можливо, більше підходить до теми сайту. Але у формі мають бути присутні не менше трьох елементів різного виду (не лише текстові поля).

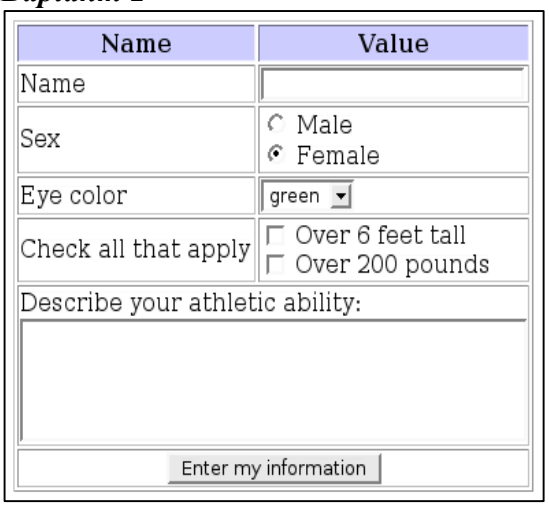
*Рекомендації до виконання*. Сторінку розбити на три частини (за допомогою таблиці або використання блоків).

– В першій частині подати зображення форми, яку необхідно розробити.
– У другій частині – безпосередньо реалізація форми, яка має збігатися за зовнішнім виглядом з зображенням.
– У третій частині – текстова інформація у вигляді деякої анкети-звіту, де буде подано введену користувачем інформацію для перевірки.

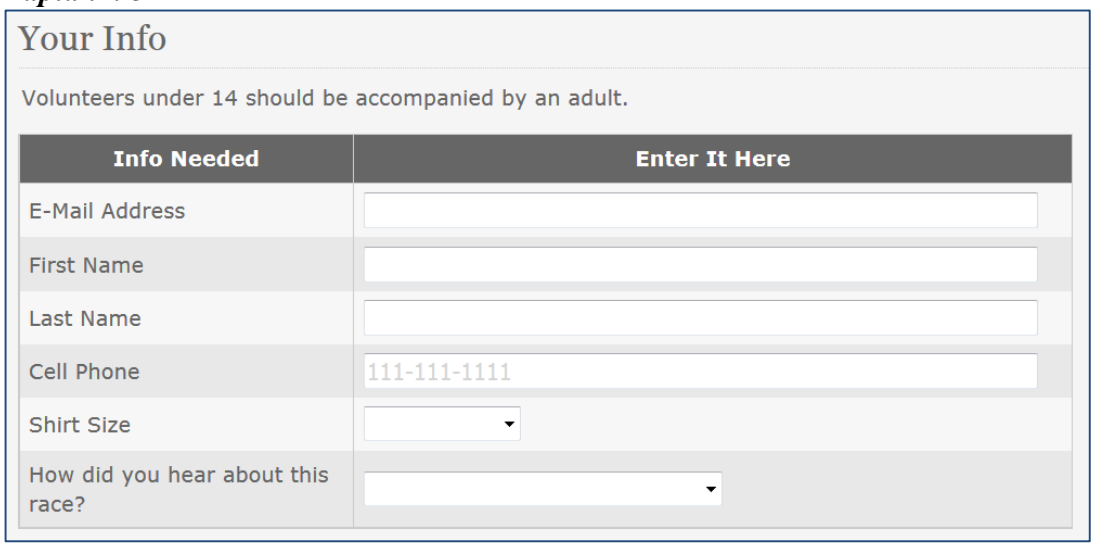
***Варіант 1***



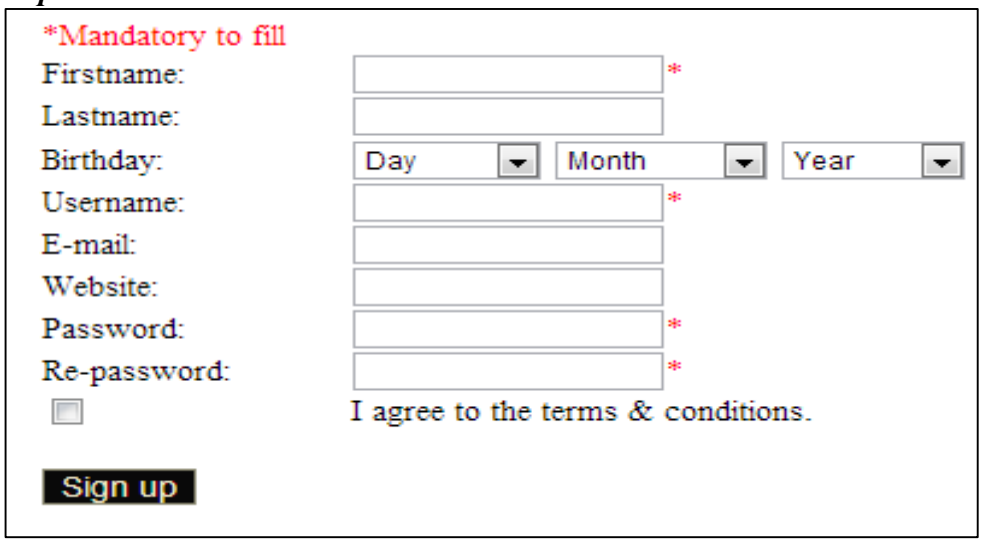

***Варіант 2***



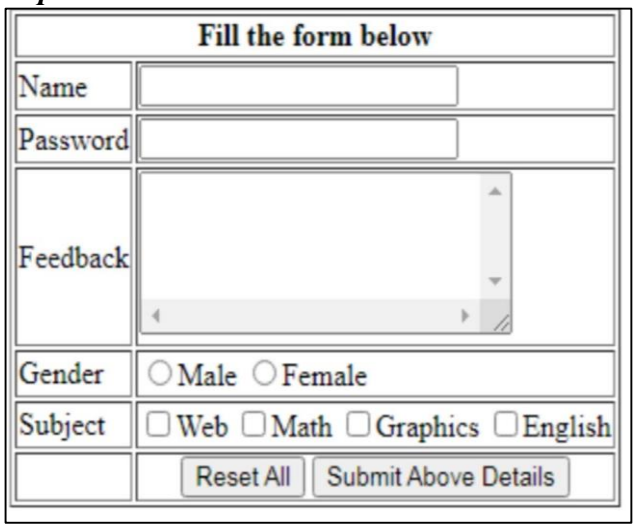
***Варіант 3***



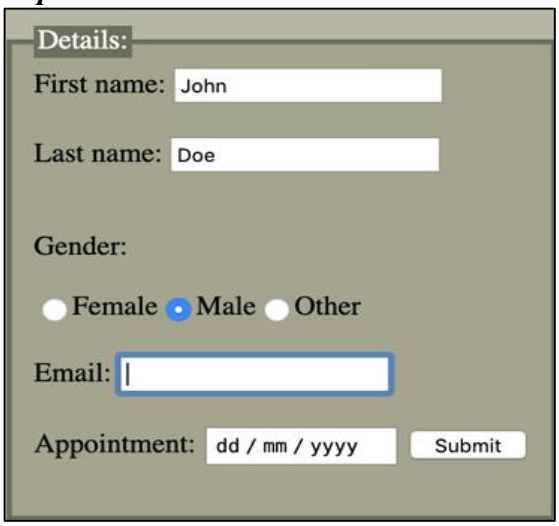

***Варіант 4***




***Варіант 5***

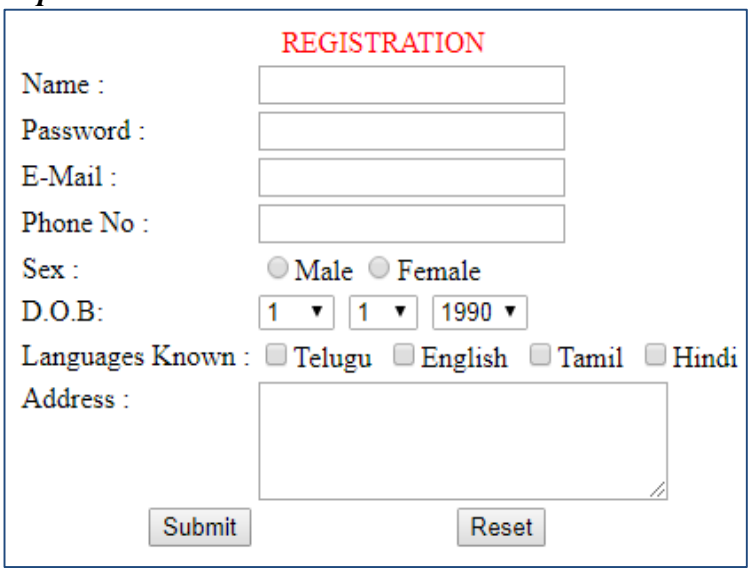



***Варіант 6***

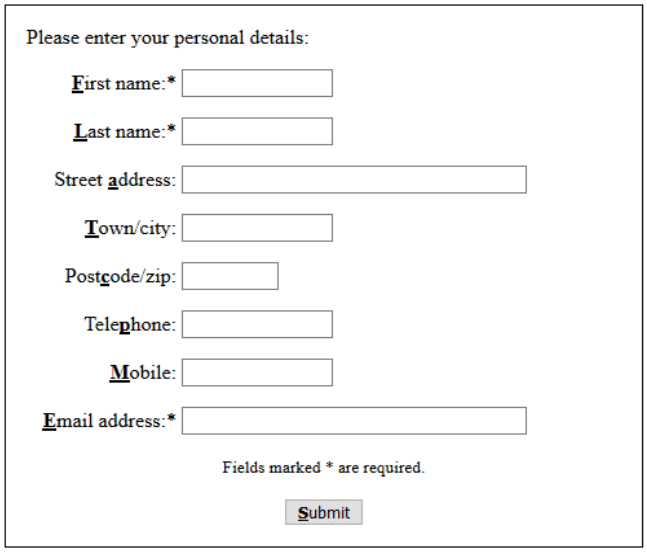


***Варіант 7***


***Варіант 8***

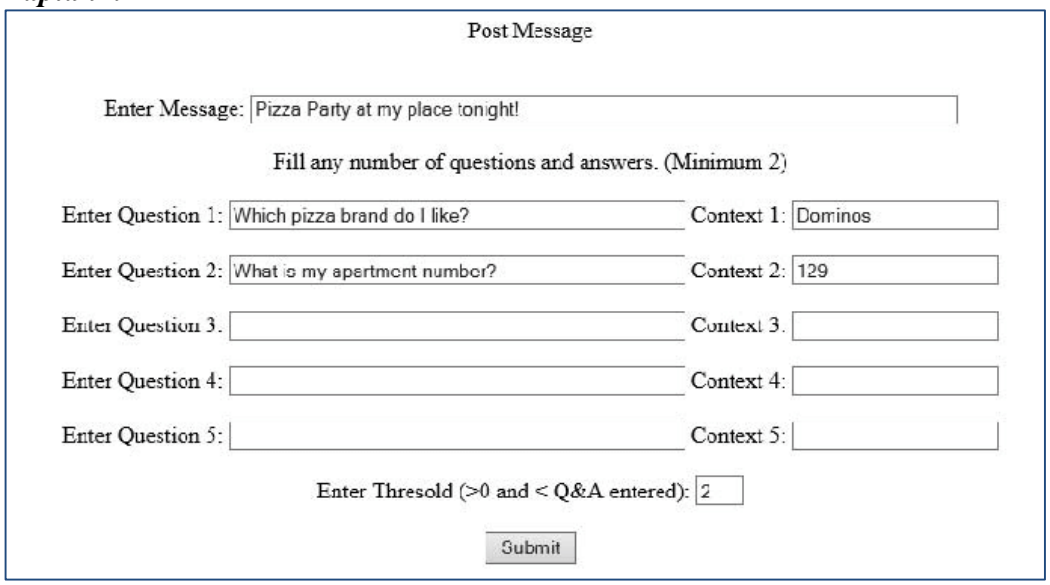



***Варіант 9***

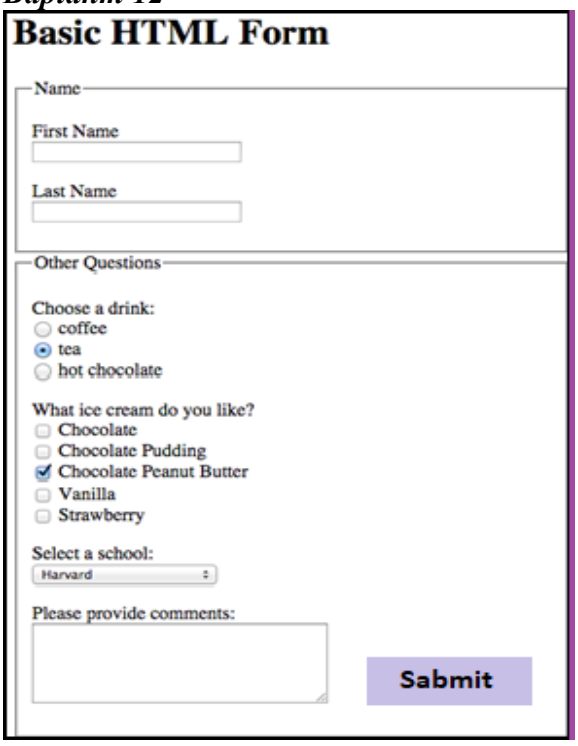



***Варіант 10***

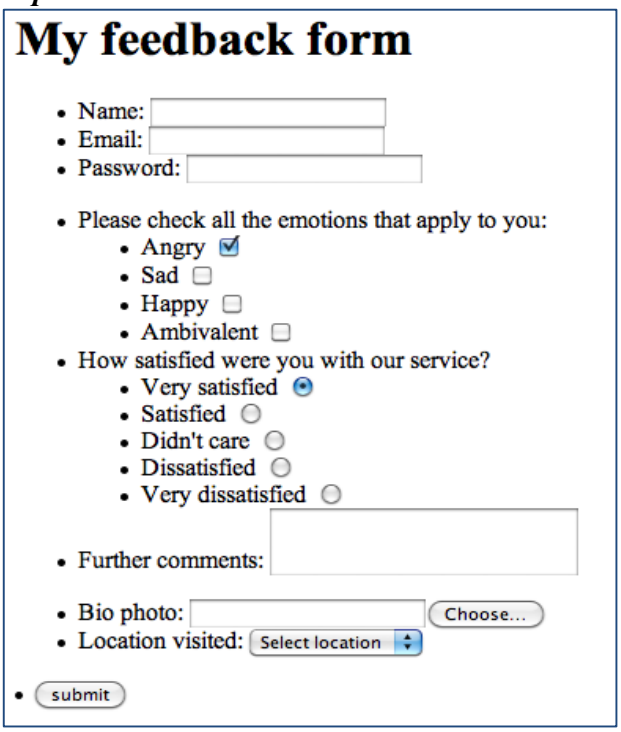

***Варіант 11***



*Варіант 12*




*Варіант 13*




*Варіант 14*


*Варіант 15*


## Додаткові джерела


1. Підручники HTML та CSS. [Електронний ресурс]: URL: https://htmlbook.at.ua/.
2. JavaScript HTML DOM Елементи. [Електронний ресурс]: URL: https://w3schoolsua.github.io/js/js_htmldom_elements.html#gsc.tab=0.
3. JavaScript Підручник. Основи веб-програмування. [Електронний ресурс]: URL: https://w3schoolsua.github.io/js/index.html#gsc.tab=0.

# 7 ОБ’ЄКТНА МОДЕЛЬ БРАУЗЕРА (BOM)

## 7.1 Основні об’єкти BOM

Модель об’єкта браузера (BOM) являє собою додаткові об’єкти, надані браузером (хост-середовищем) для роботи з усім, крім документа (рис. 7.1). Об’єкти браузера є властивостями об’єкта *window*.

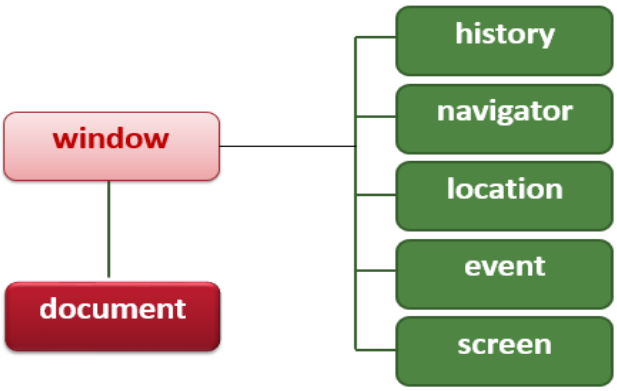



Рисунок 7.1 – Об’єкти браузера

Отже, об’єкт *window* має дві ролі:

- по-перше, це глобальний об’єкт для коду JavaScript;
- по-друге, він являє собою «вікно браузера» та надає способи для керування ним.

### 7.1.1 Об’єкт window

Кореневий об’єкт BOM – це об’єкт *window*:

– вся робота з браузером починається з вікна;
– відкриваючи нову вкладку, браузер створює новий об'єкт *window;*
– немає вікна – немає нічого (документ, елементи управління завантажуються у вікно);
– якщо в браузері відкрита одна вкладка, то створений один об'єкт *window*, якщо два, то і два об'єкти.

Як кожний об’єкт – об’єкт *window* має властивості і методи.

Основні ***властивості*:**

**`closed`** – закрито чи відкрито вікно (*true/false*);
**`defaultStatus`** – значення за замовчуванням поля статусу ;
**`document`** – посилання на об'єкт *document* цього вікна;
**`frames`** – посилання на масив фреймів цього вікна ;
**`length`** – число фреймів у вікні;
**`history`** – посилання на об'єкт *history*, який містить URL-адреси Web-сторінок, що завантажуються в це вікно;
**`location`** – встановлює або повертає URL-адреси поточної Web-сторінки;
**`innerHeight`** – внутрішня висота вікна браузера (у пікселях);

**innerWidth** – внутрішня ширина вікна браузера (у пікселях). Вікно браузера НЕ містить панелі інструментів і смуги прокручування.

***Методи*** об'єкта *window*

**open()** – створення або відкриття вікна.

```
NewWnd = window.open("URL","wndName","Features",replace)
```

*URL* – URL – адреса сторінки, яка завантажується в це вікно (або пусте);
*wndname* – задання імені вікна в атрибуті *target* тега <a>;
*features* – характеристики віконного інтерфейсу:
*toolbar* – стандартна панель інструментів + кнопки Forward, Back, переходу до домашньої сторінки і друку;
*menubar* – меню у верхній частині екрана + пункти File, Edit і View;
*location* – виводить рядок URL;
*status* – рядок стану (статусу) внизу вікна;
*scrollbar* – лінійки прокрутки, якщо документ не поміщається у вікні
*resizable* – дозволяє змінювати розмір вікна;
*width* – початкова ширина вікна в пікселях;
*height* – початкова висота вікна в пікселях.
*replace* – чи заміщає це вікно поточне у списку історії.

Приклади:

```
let myWin1 = open("something.htm");
var myWin2 = open("something.htm", "displayWindow",
     "width=400,height=300,status=no,toolbar=no,menubar=no");
```

**close**() – закриває поточне вікно;
**moveTo(X,Y)** – переміщує вікно в точку екрана, задану X і Y;
**moveBy(dX,dY)** – переміщує вікно на dX вправо і dY вниз;
**scrollTo(X,Y)** – прокручує вікно до координат X і Y;
**scrollBy** (**dX,dY)** – прокручує вікно на dX і dY;
**resizeTo(X,Y)** – змінює розмір вікна;
**resizeBy(dX,dY)** – змінює розмір вікна ;
**navigate(URL-адреса)** – завантажує Web-сторінку з адресою URL;
**print()** – друкує вміст вікна або фрейму на принтері;
**focus()** – встановлює фокус для вікна, переміщаючи на перший план;
**blur()** – видаляє фокус з вікна, переміщаючи на задній план;
**showModalDialog()** – створення модальної діалогової панелі;
**showHelp(URL)** – відображення довідкової інформації.

### 7.1.2 Об'єкт navigator

Цей об'єкт забезпечує інформацію про браузер та операційну систему.

Існує багато його властивостей, але дві найбільш широко використовуваних*: :*

**navigator.userAgent** – інформація про поточний браузер,

**navigator.platform** – інформація про платформу (може допомогти визначити, на якій платформі відкрито браузер – Windows/Linux/Mac тощо).

Інформація з об’єкта навігатора часто може вводити в оману, і її не рекомендується використовувати для визначення версій браузера, оскільки:

- різні браузери можуть використовувати однакову назву;
- дані навігатора можуть бути змінені власником браузера;
- деякі браузери помилково ідентифікують себе, щоб обійти тести сайту;
- браузери не можуть повідомляти про нові операційні системи, випущені пізніше, ніж браузер.

Існує ще ряд властивостей об’єкта *navigator*. Деякі з них наведено у прикладі.

Зауважимо, що об’єкт *window.navigator* можна записати без префікса *window*.

Приклад. Фрагмент коду:

```
<body >
    <h2>Властивості об'єкта <b><i>navigator</i></b></h2>
    <p id="propeties"></p>
<script>
   document.getElementById("propeties").innerHTML =
   "Назва програми веб-браузера: " + navigator.appName +
   "<br>Версія браузера: " + navigator.appVersion +
   "<br>Заголовок user-agent: " + navigator.userAgent +
   "<br>Платформа браузера (ОС): " + navigator.platform +
   "<br>Мова браузера: " + navigator.language +
   "<br>Файли cookie ввімкнено? " + navigator.cookieEnabled +
   "<br>Чи ввімкнено Java? " + navigator.javaEnabled() +
   "<br>Чи є браузер онлайн? " + navigator.onLine;
</script>
</body>
```

Результат:

**Властивості об'єкта *navigator***

Назва програми веб-браузера: Netscape
Версія браузера: 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36
Заголовок user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36
Платформа браузера (ОС): Win32
Мова браузера: ru-RU
Файли cookie ввімкнено? true
Чи ввімкнено Java? false
Чи є браузер онлайн? true

### 7.1.3 Об’єкт location

Об’єкт *location* дозволяє нам прочитати поточну URL-адресу і може перенаправити web-браузер на нову адресу.

Об’єкт має такі ***властивості***:

**`href`** – повертає *href* (URL) поточної сторінки;
**`.hostname`** – повертає доменне ім’я вебхостингу;
**`pathname`** – повертає шлях і назву файлу поточної сторінки;
**`protocol`** – повертає використаний веб-протокол (*http*: або *https*:).

***Методи*** об’єкта *location*:

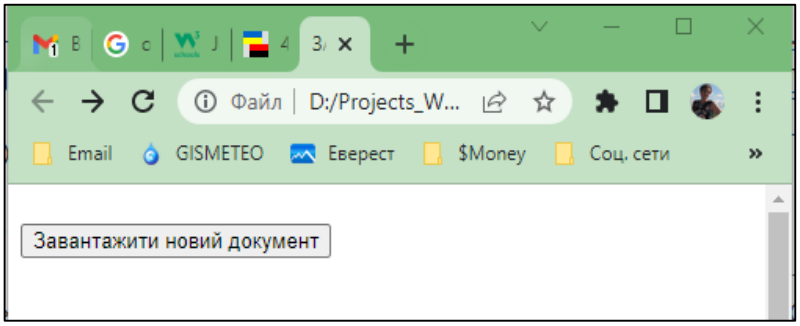
**`reload([forceget])`** – перезавантажити документ за поточним URL;
**`replace(url)`** – замінити поточний документ на документ за вказаним URL;
**`toString()`** – повертає рядкове подання URL;
**`assign`**() завантажує новий документ.

Об’єкт *window.location* можна також записати без префікса *window*.

Приклад. Фрагмент коду:

```
<body>
    <p><input type="button" value="Завантажити новий документ"
        onclick="newDoc()"></p>
    <script>
        function newDoc() {
          window.location.assign("https://www.google.com")
        }
    </script>
</body>
```

Результат:




### 7.1.4 Об’єкт *history*

Браузер, переходячи за посиланнями від сторінки до сторінки, зберігає історію цих переходів.

Об’єкт *window.history* містить історію браузерів і відповідає за кнопки «*вперед*» і «*назад*». Щоб захистити конфіденційність користувачів, існує обмеження щодо доступу JavaScript до цього об’єкта. Об’єкт має властивість length (довжина історії) і основні методи:

**`history.back()`** – те саме, що клацнути назад у веб-браузері

**history.forward()** – те саме, що клацнути вперед у веб-браузері

Приклад. Фрагмент коду:

```
<body>
    <p><input type="button" value="Назад"
onclick="goBack()"></p>
    <script>
        function goBack() {
            window.history.back()
        }
    </script>
</body>
```

Результат – за натискання на кнопку на сторінці буде викоуватись те саме, що й у разі натискання на стрілочку «вперед».

### 7.1.5 Об'єкт screen

Об'єкт window.screen містить інформацію про екран користувача:

- корисно для визначення того, що код виконується на мобільному пристрої з маленьким екраном;
- можна використовувати для збору статистичної інформації про відвідувачів (скільки відвідувачів приходило з яким екраном).

Основні властивості (справа показано результат виведення значень цих властивостей на комп'ютері авторів):

**screen.width**
**screen.height**
**screen.availWidth**
**screen.availHeight**
**screen.colorDepth**
**screen.pixelDepth**

**availWidth:** 1280
**availHeight:** 770
**availTop:** 0
**availLeft:** 0
**pixelDepth:** 24
**colorDepth:** 24
**width:** 1280
**height:** 800

Властивість **screen.colorDepth** повертає кількість бітів, використаних для відображення одного кольору.

Усі сучасні комп'ютери використовують 24- або 32-розрядне обладнання для роздільної здатності кольорів:

24 bits = 16,777,216 різних "True Colors";

32 bits = 4,294,967,296 різних "Deep Colors".

Крім того, об'єкт screen має такі методи, які працюють у повноекранному режимі або для встановлених додатків:

**Screen.lockOrientation()** – блокування орієнтації екрану;

**Screen.unlockOrientation()** – розблокувати орієнтацію екрану.

### Контрольні запитання

1. Що означає поняття «об'єктна модель браузера»?
2. Що являє собою об'єкт window? В чому його особливість?
3. Які властивості і методи має об'єкт window?
4. Які об'єкти входять в BOM?
5. Наведіть призначення і властивості основних об'єктів BOM.

## Практична робота № 7

### *Мета роботи*

Ознайомитись з основними класами об’єктної моделі браузера і навчитись отримувати властивості об’єктів BOM.

### *Завдання до виконання роботи*

Використовуючи можливості класів об’єктної моделі браузера, представити таку інформацію:

- надати всю інформацію про браузер: назва браузера і кодову назву браузера і т.д.;
- дізнатися, чи включена можливість використання cookie в браузері;
- дізнатися про версію браузера;
- дізнатися про платформу, під яку скомпільовано браузер;
- перевірити, чи дозволена Java в браузері;
- отримати шлях до завантаженого документу;
- дізнатися кількість відвіданих URL, що зберігаються в списку;
- дізнатися ім'я домену завантаженого документа та назву протоколу;
- визначити ширину і висоту екрана (включаючи і виключаючи такі елементи інтерфейсу як смуга прокрутки, рядок стану, панель інструментів і т. д.);
- дізнатися глибину кольору.

Ця інформація має бути подана різними способами (з використанням різних елементів керування) відповідно до індивідуального завдання:

| *Варіант* | *Завдання* |
|---|---|
| 1, 5, 9, 13 | Пункти поданого списку – комбінований список. Під час наведення курсора миші на будь-який пункт з’являється вікно-підказка з відповідною інформацією. Поруч зі списком розташувати текстове поле, в якому у разі вибору певного пункту відображатиметься відповідна інформація. |
| 2, 6, 10, 14 | Пункти поданого списку – прапорці (кнопки з незалежною фіксацією. Додатково під списком розмістити кнопку ОК. Після встановлення певних прапорців сформована інформація за натискання на кнопку ОК виводиться у документ (можна в окремому вікні). |
| 3, 7, 11, 15 | Пункти поданого списку – звичайний перелік (маркований список). За наведення курсора миші на будь-який пункт з’являється вікно-підказка з відповідною інформацією. Поруч зі списком розташувати текстове поле, в якому у разі вибору певного пункту відображатиметься відповідна інформація. |
| 4, 8, 12, 16 | Пункти поданого списку – селекторні кнопки (кнопки з залежною фіксацією). Додатково під списком розмістити кнопку ОК. Після встановлення певних прапорців сформована інформація за натискання на кнопку ОК виводиться у документ (можна в окремому вікні). |

## ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ


1. HTML. Living Standart. https://html.spec.whatwg.org/
2. W3schoolsUA.українською. https://w3schoolsua.github.io/#gsc.tab=0.
3. JavaScript Підручник. Основи веб-програмування. [Електронний ресурс]: URL: https://w3schoolsua.github.io/js/index.html#gsc.tab=0.
4. Мельник Р. А. Програмування веб-застосувань (фронт-енд та бек-енд). Львів: Львівська політехніка, 2018. 248 с.
5. Бородкіна І. Л., Бородкін Г. О. WEB-технології та WEB-дизайн: застосування мови HTML для створення електронних ресурсів. К. : Ліра-К, 2020. 212 с.
6. Руденко В. Д. Речич Н. В., Потієнко В. О. Інформатика. Профільний рівень. Харків : Ранок, 2020. 256 с.
7. Підручники HTML і CSS. [Електронний ресурс]: URL: https://htmlbook.at.ua/.
8. HTML5 Семантична верстка. [Електронний ресурс]: URL: https://avivi.pro/ua/blog/html5-semanticheskaya-vyerstka/.
9. Довідник CSS атрибутів. [Електронний ресурс]: URL: https://html-css.co.ua/dovidnuk-css-atrubytiv/.
10. Український веб-довідник. [Електронний ресурс]: URL: https://css.in.ua/.
11. METANIT.COM. [Електронний ресурс]: URL: https://metanit.com/web/html5/.
12. Безкоштовний довідник по CSS. [Електронний ресурс]: URL: https://cssreference.io.
13. Блочна модель CSS. [Електронний ресурс]: URL: https://www.w3schools.com/css/css_boxmodel.asp#.
14. Блочна модель CSS. [Електронний ресурс]: URL: https://webportal.com.ua/box-model-css/.
15. Псевдокласи CSS. [Електронний ресурс]: URL: https://html-css.co.ua/self-css/psevdoklasi/.
16. CSS: псевдоелементи і псевдокласи. [Електронний ресурс]: URL: https://hi-news.pp.ua/kompyuteri/6296-css-psevdoelementi-psevdoklasi.html.
17. Красиві css3 ефекти, які легко реалізувати. [Електронний ресурс]: URL: http://yoip.com.ua/krasivi-css3-efekti-yaki-legko-realizuvati/.
18. CSS-анімації. [Електронний ресурс]: URL: https://uk.javascript.info/css-animations.
19. Застосування анімацій CSS. [Електронний ресурс]: URL: https://webdoky.org/uk/docs/Web/CSS/CSS_Animations/Using_CSS_animations/#nalashtuvannia-animatsii.
20. Використання Flexbox в CSS3 для адаптивного дизайну. [Електронний ресурс]: URL: https://sebweo.com/vikoristannya-flexbox-v-css3-dlya-adaptivnogo-dizajnu/.

21. Як працює CSS Flexbox. [Електронний ресурс]: URL: https://petrov.net.ua/how-css-flexbox-works/.
22. Шпаргалка по Flexbox CSS. [Електронний ресурс]: URL: https://tpverstak.ru/flex-cheatsheet/.
23. A Complete Guide to Flexbox. [Електронний ресурс]: URL: https://css-tricks.com/snippets/css/a-guide-to-flexbox/.
24. Знайомство з CSS Grid Layout. [Електронний ресурс]: URL: https://css.in.ua/article/znayomstvo-z-css-grid-layout_12
25. Grid в CSS: повний посібник та довідник з верстки. [Електронний ресурс]: URL: https://highload.today/uk/grid-v-css-povnij-posibnik-ta-dovidnik-z-verstki/.
26. 21 кращий конструктор сайтів: повний огляд. [Електронний ресурс]: URL: https://hostiq.ua/blog/ukr/site-builders/.
27. Топ-6 кращих конструкторів сайтів у 2022 році. [Електронний ресурс]: URL: https://cityhost.ua/uk/blog/top-6-luchshih-konstruktorov-saytov-v-2022-godu.html.
28. Безкоштовні адаптивні шаблони веб-сайтів. [Електронний ресурс]: URL: https://www.templatemonster.com/ua/free-templates/website-template.html.
29. Галерея дизайнів. [Електронний ресурс]: URL: https://horoshop.ua/ua/design/.
30. FIGMA. [Електронний ресурс]: URL: https://www.figma.com/design/.
31. Bootstrap. [Електронний ресурс]: URL: https://getbootstrap.com/.
32. 8 Кращих БЕЗКОШТОВНИХ Хостингів у 2023 році. [Електронний ресурс]: URL: https://www.websiteplanet.com/uk/blog/.
33. Сучасний підручник з Javascript. [Електронний ресурс]: URL: https://uk.javascript.info/.
34. JavaScript Cookbook: Programming the Web 3rd Edition. Adam D. Scott, Matthew MacDonald, Shelley Powers. O'Reilly Media, Inc..August 24, 2021. ISBN: 9781492055709.
35. PC VECTOR. [Електронний ресурс]: URL: https://pcvector.net/.
36. Об'єктно-орієнтоване програмування. [Електронний ресурс]: URL: http://xn--80adth0aefm3i.xn--j1amh/%D0%9E%D0%9E%D0%9F.
37. Стандартні вбудовані об'єкти. [Електронний ресурс]: URL: https://webdoky.org/uk/docs/Web/JavaScript/Reference/Global_Objects/#standartni-obiekty-za-katehoriiamy.

**Додаток А**

**Приклад оформлення титульного аркуша звіту**

Міністерство освіти і науки України
Вінницький національний технічний університет
Інститут інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

**Звіт**
**з практичної роботи №1**
**«Створення web-сайту засобами мови HTML»**

**Варіант № 5**

Розробив студент гр. 1 БС-23 б
_____ Холодов І. В.

Практичну роботу захищено
з оцінкою _____

Перевірив ст. викл. каф. ЗІ
_____ Каплун В. А.
_____ 2023 р.

ВНТУ 2023

## Додаток Б

### Налагодження програми в браузері

** Інформацію взято з джерела: https://uk.javascript.info/debugging-chrome.*

Налагодження – це процес пошуку і виправлення помилок в скрипті. Усі сучасні браузери і більшість інших середовищ розробки підтримують інструменти налагодження – спеціальний графічний інтерфейс, який значно спрощує налагодження. Він також дозволяє покроково відслідковувати, що саме відбувається в коді.

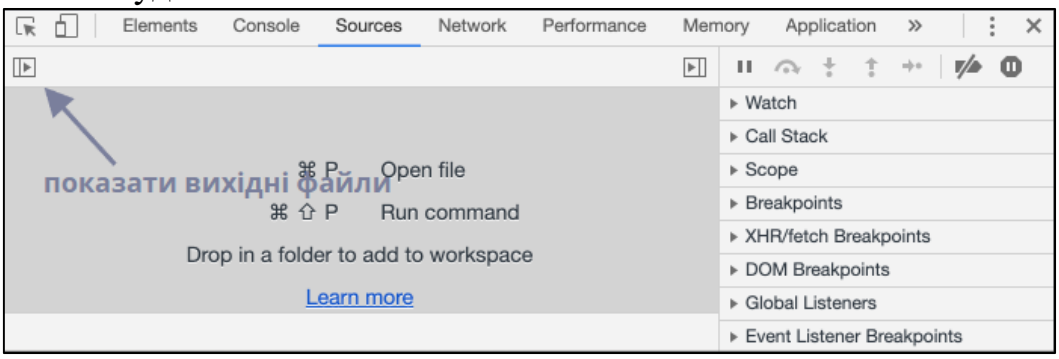
Ми будемо використовувати браузер Chrome, оскільки у нього достатньо можливостей для налагодження. В більшості інших браузерів процес буде схожим.

**Вкладка «Sources» («вихідний код»)**

Ваш браузер Chrome може бути іншої версії – він може виглядати інакше, але різниця не буде суттєвою.

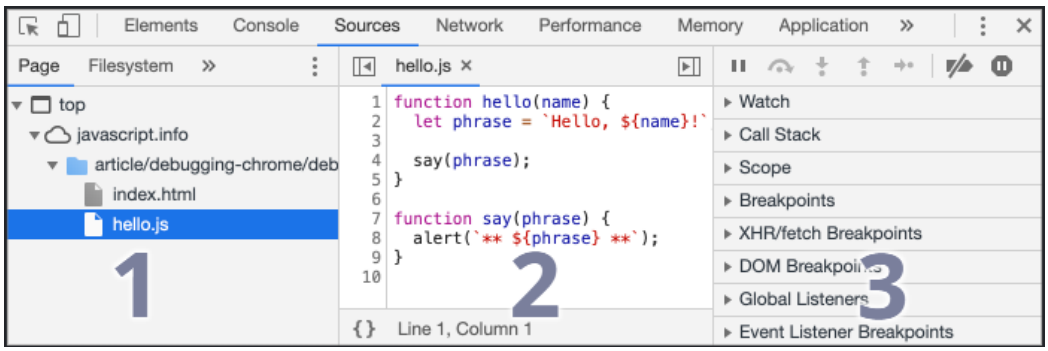
- В браузері Chrome, відкрийте тестову сторінку (https://uk.javascript.info/article/debugging-chrome/debugging/index.html)
- Відкрийте інструменти розробника, натиснувши клавішу F12 (або Cmd+Opt+I на Mac).
- Виберіть вкладку **Sources**.

У вас буде схоже вікно:




Кнопка-перемикач ліворуч відкриває панель з файлами.

Натисніть на неї і виберіть файл **hello.js.** Ось як буде виглядати вкладка **Sources**:



Цей інтерфейс складається з трьох частин:

1. На панелі **Навігатор** файлів (File Navigator) показано файли HTML, JavaScript, CSS та інші файли, включно із зображеннями, які використовуються на сторінці. Також тут можуть бути файли від розширень Chrome.
2. Панель **Редагування коду** (Code Editor) показує вихідний код.
3. Панель **Налагодження JavaScript** (JavaScript Debugging) використовується для налагодження, ми повернемося до цього пізніше.

Можна знову натиснути на ту саму кнопку, закрити панель і звільнити місце.

### Консоль

Якщо натиснути клавішу **Esc**, в нижній частині екрана відкриється консоль. Туди можна вводити команди і їх виконувати, натиснувши клавішу **Enter**.

Нижче показується результат виконання команд.

Наприклад, результатом 1+2 буде 3, а ось інструкція

*hello("debugger")*

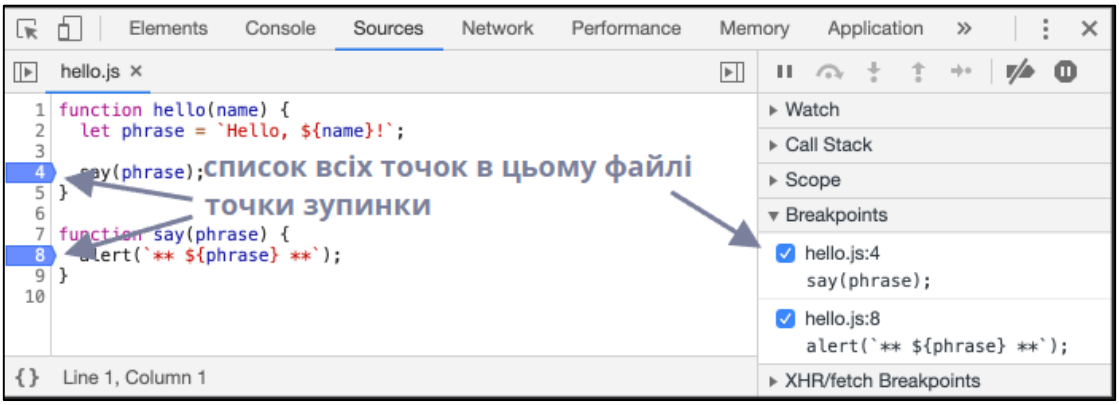
нічого не повертає, тому результат буде *undefined*:


### Точки зупинки (breakpoints)

Давайте розберемося, як працює код на тестовій сторінці. В файлі *hello.js*, натисніть на рядок номер 4 (на саму цифру, а не на код).

Вітаємо! Ви поставили точку зупинки. Поставте також точку зупинки на 8 рядку.

Номери рядків мають стати синього кольору. Ось що має вийти:




Точка зупинки – це місце в коді, де налагоджувач автоматично (тимчасово) зупинить виконання JavaScript.

Поки виконання (тимчасово) зупинене, ми можемо переглядати поточні значення змінних, виконувати команди в консолі тощо. Інакше кажучи, можемо налагоджувати (розробники інколи кажуть «дебажити», від слова «debug»).

В правій частині панелі видно всі точки зупинки. Коли виставлено багато таких точок, та ще й в різних файлах, цей список дозволяє ефективно ними керувати:

- швидко переміщатися до будь-якої точки зупинки в коді – потрібно клацнути по ній в правій частині панелі;
- тимчасово вимкнути точку зупинки, знявши виділення;
- видалити точку – потрібно клацнувши по ній правою кнопкою миші й вибрати «Remove breakpoint» (видалити точку зупинки)
- …тощо.

> ***Умовні точки зупинки.*** Можна задати так звану умовну точку зупинки – клацніть правою кнопкою миші по номеру рядка в коді, виберіть пункт «Edit breakpoint…» і пропишіть умову. Коли ця умова виконається, то виконання коду (тимчасово) зупиниться в цій точці зупинки.
>
> Цей метод використовується, коли потрібно (тимчасово) зупинити виконання коду під час специфічних значень змінних або параметрів функції.

### Команда debugger

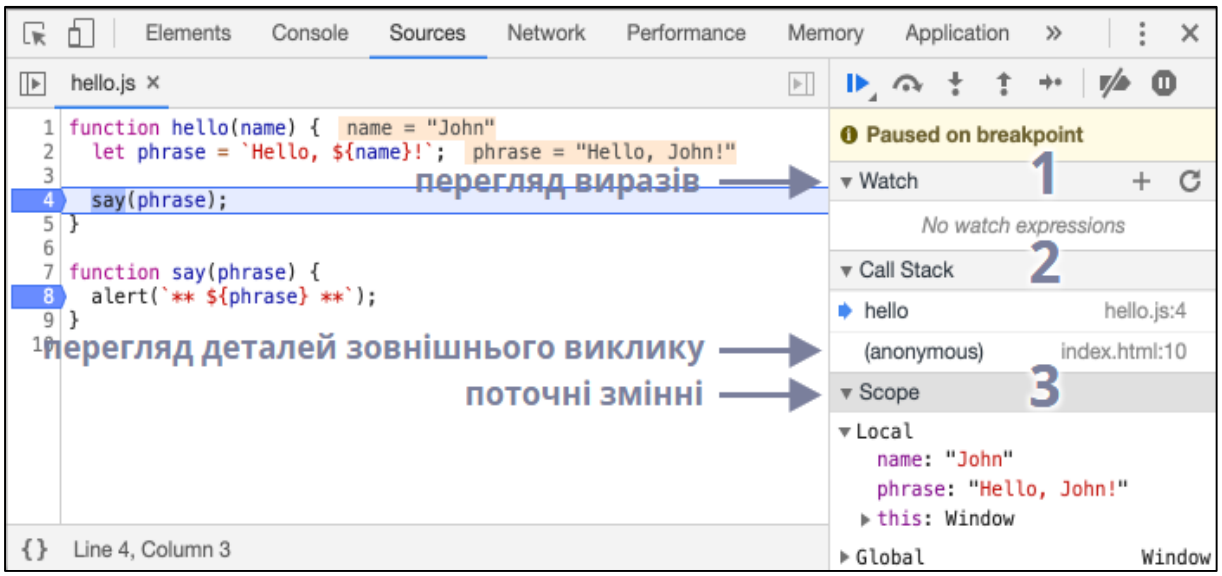
Виконання коду також можна (тимчасово) зупиняти командою debugger прямо всередині коду, ось так:

```
function hello(name) {
  let phrase = `Привіт, ${name}!`;
  debugger;  // <-- тут зупиниться налагоджувач
  say(phrase);
}
```

Ця команда працює лише тоді, коли інструменти розробника відкриті, інакше браузер її ігнорує.

У нашому прикладі, функція *hello()* викликається під час завантаження сторінки, отже, найшвидшим способом активувати налагоджувач (після того як ми поставили точку зупинки) — це перезавантажити сторінку. Тому просто натисніть F5 (Windows, Linux) чи Cmd+R (на Mac).

Оскільки ми поставили точку зупинки, виконання коду (тимчасово) зупиниться на 4-му рядку:




Щоб зрозуміти, що відбувається в коді, натисніть на стрілки справа. Можна виділити три основні блоки:

1. **Watch** показує поточні значення виразів.

Можете натиснути на + і ввести якийсь вираз. В процесі виконання налагоджувач автоматично перераховуватиме і показуватиме його значення.

2. **Call Stack** показує послідовність викликів функцій.

В нашому прикладі налагоджувач (тимчасово) зупинив виконання коду всередині функції *hello()*, яка була викликана з файлу *index.html* (там немає функції, тому виклик «anonymous» – анонімний).

Натиснувши на елемент списку (наприклад, на «anonymous»), налагоджувач перейде до відповідного коду, де було здійснено виклик.

3. **Scope** показує поточні змінні.

В **Local** показуються локальні змінні функції, а їх значення підсвічуються в вихідному коді.

В **Global** показуються глобальні змінні (тобто ті, які оголошені поза функціями).

Зверніть увагу, що під час зміни викликів функцій (з блока «Call Stack»), поточні змінні теж змінюються. Тут ще є ключове слово this, поки що не звертайте на нього уваги — ми вивчимо його пізніше.

### Відстежування виконання коду

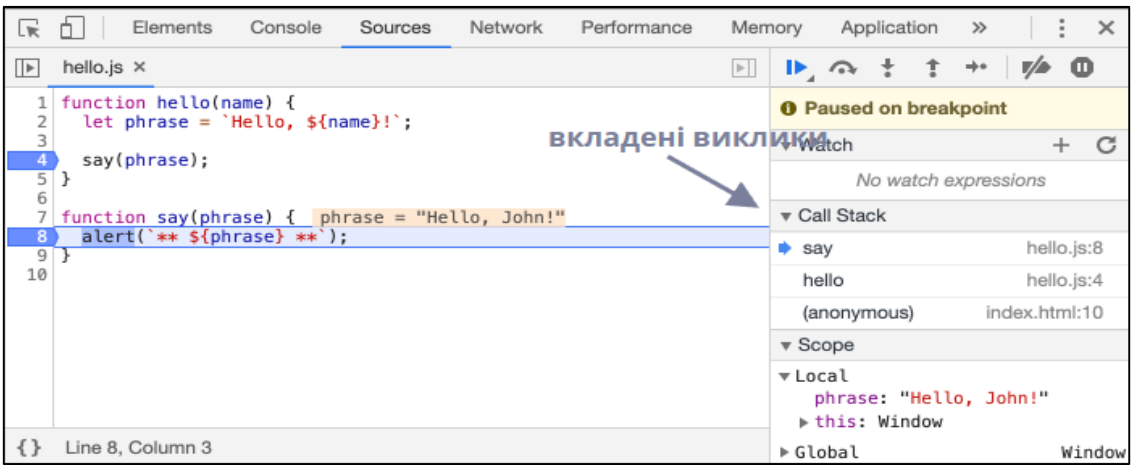
Настав час відстежувати (trace) скрипт.

Для цього є декілька кнопок, які знаходяться на панелі зверху праворуч. Давайте їх розглянемо.

– **Resume**: продовжити виконання. Швидка клавіша: **F8**.

Відновлює виконання коду. Якщо більше немає точок зупинок, налагоджувач завершить свою роботу, а код буде виконуватися далі.

Ось що ми побачимо, коли натиснемо на цю кнопку:




Виконання коду відновилося, дійшло до іншої точки зупинки, всередині *say()* і налагоджувач знову (тимчасово) зупинив виконання. Зверніть увагу на вкладку **Call Stack** праворуч: в списку з'явився ще один виклик. Ми тепер всередині функції *say()*.

– **Step**: виконати наступну команду. Швидка клавіша: **F9**.

Якщо ми натиснемо на неї – виконається функція *alert*. Натискаючи цю кнопку раз за разом, всі вирази будуть виконуватися покроково.

– **Step over**: виконати наступну команду, не заходячи в функцію. Швидка клавіша: **F10**.

Подібна до попередньої команди «Step», проте працює дещо по-іншому, якщо наступний вираз – виклик функції (не вбудована, як *alert*, а наша власна функція).

Команда «Step» зайде в функцію і зупиниться на її першому рядку, тоді як «Step over» виконає всі вирази, які є в цій функції (включно з викликами вкладених функцій, такі як *alert()*).

Виконання (тимчасово) зупиниться на наступному рядку, коли завершиться функція. Це зручно, коли ми не хочемо досліджувати, що відбувається всередині функції.

– **Step into**: зробити крок. Швидка клавіша: **F11.**

Подібна до «Step», але працює інакше у випадку викликів асинхронних функцій. Якщо ви тільки почали вчити JavaScript, тоді можете проігнорувати цю різницю, тому що ми поки що не вчили асинхронних викликів.

На майбутнє просто майте на увазі, що команда «Step» ігнорує асинхронні дії, такі як setTimeout (відкладений виклик функції), які виконуються пізніше. Команда «Step into» заходить в їхній код, і очікує на них, якщо потрібно. Можете поглянути в документацію DevTools, щоб побачити як це відбувається.

– **Step out**: продовжити виконання до завершення поточної функції. Швидка клавіша: **Shift+F11**.

Виконання коду відновиться і (тимчасово) зупинитися на останньому рядку поточної функції. Це зручно, коли ми випадково натиснули кнопку , зайшовши у вкладений виклик, і хочемо якнайшвидше завершити його.

– активувати/деактивувати всі точки зупинки.

Ця кнопка не впливає на виконання коду, вона лише дозволяє масово увімкнути/вимкнути точки зупинки.

Коли ця кнопка активна і відкрито інструменти розробника, тоді скрипт автоматично (тимчасово) зупинить виконання, якщо трапиться якась помилка. Ми зможемо проаналізувати змінні й дослідити, що пішло не так. Отже, якщо наш скрипт аварійно завершує роботу, ми можемо відкрити інструменти розробника, активувати цю опцію і перезавантажити сторінку, щоб побачити де і за яких умов скрипт «вмирає», і які в цього деталі.

**Логування**

Щоб вивести щось в консоль з нашого коду, існує спеціальна функція `console.log`. Наприклад, така інструкція виведе в консоль числа від 0 до 4:

```
```js run
// відкрийте консоль, щоб побачити
for (let i = 0; i < 5; i++) {
  console.log("число,", i);
}
```

Звичайні користувачі не бачитимуть цієї інформації – вона в консолі. Щоб побачити її, відкрийте інструменти розробника і перейдіть на вкладку «Console», або натисніть клавішу **Esc**, якщо ви на іншій вкладці: це відкриє консоль знизу.

Якщо в нас достатньо логів в нашому коді, ми зможемо побачити що відбувається з нашими записами без допомоги налагоджувача.

**Підсумки**

Як бачимо, є три способи (тимчасово) зупинити виконання скрипту:

1. Точка зупинки.
2. Інструкція debugger.
3. Помилка (якщо активовано кнопку в інструментах розробника).

Коли виконання (тимчасово) зупинене, ми можемо налагоджувати (інколи кажуть «дебажити») – досліджувати змінні й відстежувати виконання коду, щоб побачити, що пішло не так.

В інструментах розробника набагато більше опцій, ніж ми розглянули тут. Всю інформацію про інструменти розробника браузера Chrome можна прочитати в їхній офіційній документації (англійською).

Інформації з цього розділу достатньо, щоб почати налагодження, проте пізніше, особливо якщо ви тісно працюватимете з браузером, не полінуйтеся прочитати про розширені можливості інструментів розробника.

Можна натискати на різні місця в інструментах розробника, і побачити що відбувається. Це, напевно, найшвидший спосіб ознайомитися з функціоналом інструментів розробника. Не забувайте про клацання правою кнопкою миші та контекстні меню!
```

*Навчальне електронне видання*
*комбінованого використання.*
*Можна використовувати в локальному та мережному режимах*

**Валентина Аполінаріївна Каплун**
**Микита Сергійович Ціхоцький**
**Віталій Володимирович Лукічов**

# Основи web-програмування. Теорія і практика

**Навчальний посібник**

Рукопис оформила *В. Каплун*

Редактор *Т. Старічек*

Оригінал-макет підготувала *Т. Старічек*

Підписано до видання 20.09.2023 р.
Гарнітура Times New Roman.
Зам. № Р2023-104.

Видавець та виготовлювач
Вінницький національний технічний університет,
Редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95, м. Вінниця, 21021.
Тел. (0432) 65-18-06.
**press.vntu.edu.ua**;
*E-mail*: irvc.ed.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.