МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

М. С. Максимов

«Web-програмування»

Навчально-методичний посібник
для студентів спеціальності «Економіка»
освітньої програми «Економічна кібернетика»

Харків
2024

УДК 004.55
С88

Рецензенти

Стяглик Н. І., завідувач кафедри інформаційних технологій та математичного моделювання каразінського банківського інституту Харківського національного університету імені В.Н. Каразіна, кандидат педагогічних наук;

Лубенець С. В., доцент кафедри економічної кібернетики та прикладної економіки Харківського національного університету імені В.Н. Каразіна, кандидат технічних наук

Максимов М.С.

С88 **Web-програмування**: Навчально-методичний посібник для студентів спеціальності «Економіка» освітньої програми «Економічна кібернетика» / М. С. Максимов. – Харків : 2024. – 55 с.

У навчальному посібнику розглянуті основи верстки сторінок із використанням мов розмітки HTML та CSS, а також використання мови програмування JavaScript як основної мови програмування користувацьких інтерфейсів в Web-системах. Розглянуто питання базових алгоритмів, динамічної роботи із елементами моделі DOM, а також роботи із даними, в тому числі в форматі JSON Призначено для студентів бакалаврського рівня підготовки всіх спеціальності «Економіка» освітньої програми «Економічна кібернетика», як науково-методичний посібник курсу Web-програмування.

УДК 004.55

Частина 1. Створення сторінки із використанням HTML та CSS

Розділ 1. Перша сторінка.

1.1. Основні визначення

Інтернет – це глобальна комунікаційна мережа. Інтернет має клієнт-серверну архітектуру, тобто складається з двох типів пристроїв: клієнтів та серверів.

Клієнт – пристрій або програма, які надсилають запити та отримують відповіді на них.

Сервер – пристрій або програма, які отримують запити та відправляють відповіді на них.

Інтернет є один з основних факторів сучасного обміну інформацією та ведення бізнесу. Ключовою одиницею Інтернету як засобу доступу до інформації є сайт.

Сайтом ми будемо назви вату сукупність Web-сторінок, які мають загальну навігацію та зміст та розміщені в Інтернеті під певним доменним іменем.

Web-сторінка – це документ гіпертексту, створений із використанням мов розмітки HTML та CSS, а також із використанням клієнтської мови програмування JavaScript.

Гіпертекст – це текст, що має в собі гіперпосилання.

Гіперпосилання (або просто посилання) – технологічне рішення, що дозволяє переходити від однієї Web-сторінки до іншої.

Доменне ім'я – символьне ім'я за яким користувач Інтернету може отримати головну сторінку відповідного сайту.

Браузер – клієнтське програмне забезпечення, що дозволяє переглядати Web-сторінки

1.2. Сайт

Сайт - загалом за своїми принципами аналогічний книзі. Мова розмітки HTML (Hypertext markup language) - це завдання структури сторінок, а CSS (Cascading Style Sheets) - це оформлення книги: вирівнювання, колір, відступи, фон. Все, що ви можете уявити (і бачили), що оформляє текст, реалізується через CSS. Також через CSS реалізована можливість будувати блоки один відносно іншого (верстати).

В цьому розділі ми будемо вчитися оформляти текст, який зверстано за допомогою HTML.

Але тут виникає питання... А як же сказати браузеру - що ось такий елемент потрібно таким ось чином оформити. Наприклад, заголовок зробити більшим, по центру і червоним з котиками на фоні!

1.3. Передісторія

HTML спочатку використовувався для оформлення текстових документів, тобто він не був призначений для верстки елементів один відносно одного, крім як за принципами тексту. Першою технологією, яка дозволила верстати сторінку, була технологія верстки таблицями. Це була достатньо ефективна технологія, яка дозволяла однозначно позиціонувати елементи. Але при ускладненні структури сайтів ця технологія заплутувала розробника і технологічно ця гілка виявилася тупиковою.

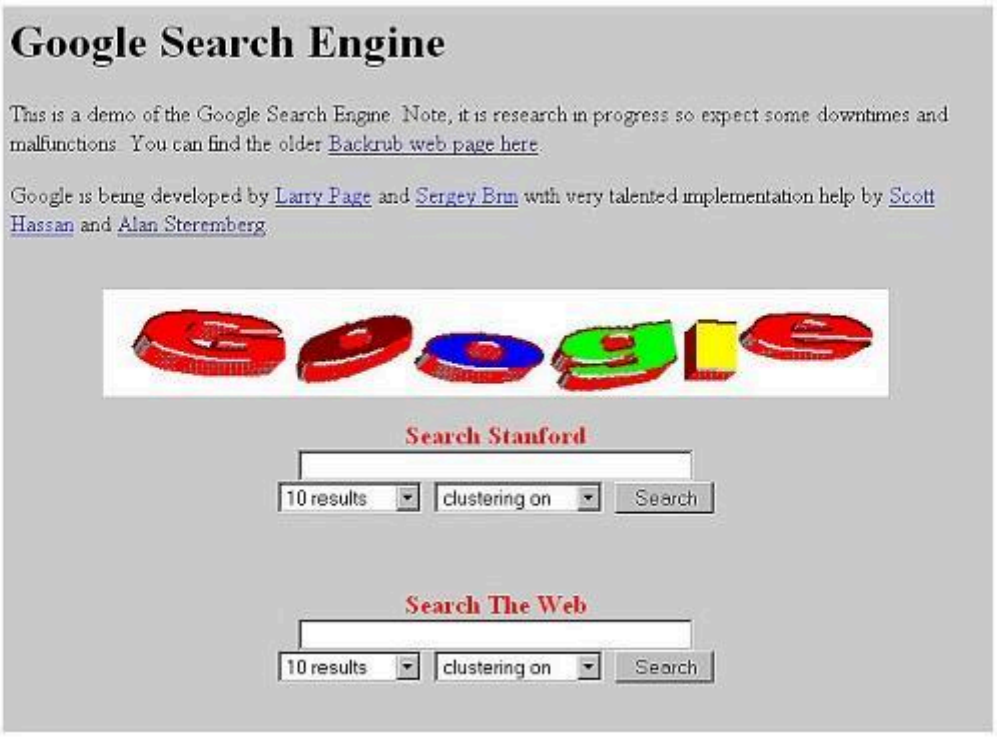

Рис. 1. Сторінка пошукача Google в 1996 році.

На зміну верстці таблиць прийшла верстка блоками з використанням властивості обтікання (float). Саме по собі це властивість спочатку призначалася для реалізації обтікання текстом картинок.

Технологія верстка блоками не була однозначно кращою за технологію верстки таблицями, і тривалий час (кілька років) у спільнотах верстальників точилися гарячі дискусії - як верстати: таблицями чи блоками. Крапку в цій історії (у битві технологій) поставив Google, який вирішив понижувати сайти, зверстані таблицями, у пошуковій видачі.

З тих пір з'явилося ще кілька технологій, які зробили верстку за допомогою обтікання (float) не актуальною. На 2024 рік основною технологією верстки є технологія flex-box.

1.4. Базові теги HTML

Основним елементом оформлення гіпертексту є тег (tag), мітка, що визначає відображення тексту в браузері.

Теги можуть бути **відкриваючими** (символ «<», назва тега, символ «>») та **закриваючими** (символ «<», символ «/», назва тега, символ «>»)

Теги можуть бути **парними:** такими що мають і відкриваючий і закриваючий тег, наприклад *Текст, виділений курсивом* який змінює оформлення тексту, що розміщений між відкриваючим за кариваючим тегом на виділення курсивом. Та **непарними –** такими, у яких є тільки відкриваючий тег немає закриваючого, зазвичай вони позначають певний елемент сторінки, наприклад <img> - картинка,
 - розрив рядка, <hr> - горизонтальна лінія.

Для простоти розуміння оформлення HTML-документу ми будемо розглядати елементи HTML поступово, приблизно так, як розвивалася ця мова розмітки.

В першій версії HTML було невелика кількість, які визначали технічні елементи оформлення гіпертексту. Перелік тегів, який буде описуватися на є вичерпним, проте дозволяє сформувати загальне розуміння використання тегів. Для простоти розуміння виділимо наступні типи найдавніших тегів:

- **Теги структури документу** - визначають структуру документу.
- **Теги для оформлення тексту:**

- **Рядкові теги –** не мають самостійної ширини та висоти, змінюють оформлення тексту, що лежить між відкриваючим та закриваючим тегом, продовжують рядок. Наприклад: (виділення курсивом)
- **Блокові теги –** мають власні ширину та висоту, займають весь доступний об'єм на сторінці, переходять на новий рядок. Наприклад (абзац)
- **Рядково-блокові –** мають ширину та висоту, проте продовжуть рядок. Наприклад <img> (відображення картинки).

Теги структури документу:

- <html></html> – парний тег, що визначає початок та кінець HTML документа.
- <head></head> – парний тег, визначає місце для службових даних документу.
- <title></title > – парний тег визначає заголовок веб-сторінки, який відображається в браузері.
- <body></body > – парний тег визначає місце розміщення основного змісту web-сторінки.

Розміщення тегів структури один відносно іншого.

```
<html>
<head><title>Заголовок сторінки</title></head>
<body>
Основний зміст сторінки.
</body>
</html>
```

Рядкові теги.

- **** – виділення тексту жирним шрифтом.
- **** – виділення тексту курсивом.
- **<u></u>** – виділення тексту підкреслюванням.
- **<a>** – визначає посилання на іншу веб-сторінку або документ.

Блочні теги – теги що мають ширину та висоту та заповнюють весь доступний об'єм сторінки.

- <h1></h1>, <h2></h2>, <h3></h3>, <h4></h4>, <h5></h5>, <h6></h6>,– виділення заголовків різного рівня (h1 – найбільший, h6 - найменший):
- ** ** – виділення абзацу тексту.
- ** ** – виділення змістовного блоку.
- **<li>** – виділяє елемент списку.

Елемент списку має бути вкладений в тег списку:

- **
** – маркерований список
- **
** – нумерований список.

Наприклад:

```
<ul>
  <li>перший елемент маркерованого списку</li>
  <li>другий елемент маркерованого списку</li>
</ul>
```

1.5. Робоче місце

Для написання коду на мовах розмітки використаємо текстовий редактор Microsoft Visual Studio Code (Рис. 2.) скачаємо даний текстовий редактор та встановимо його на власний ком'ютер.

Далі – створимо папку із використанням стандартних інструментів вашої операційної системи (в нашому прикладі ми будемо випростовувати папку Web-programming.

Після створення папки – відкриємо її як робочий простір в Microsoft Visual Studio Code (рис. 3. та рис. 4.), а далі – створимо файл, в який ми плануємо писати код. В нашому прикладі ми створимо файл index.html (рис. 5.).

Після створення файла – вставимо HTML-код в робочу область текстового редактора (рис. 6.), запустим даний код на виконання (рис. 7.) та зможемо отримати результат (рис. 8.)

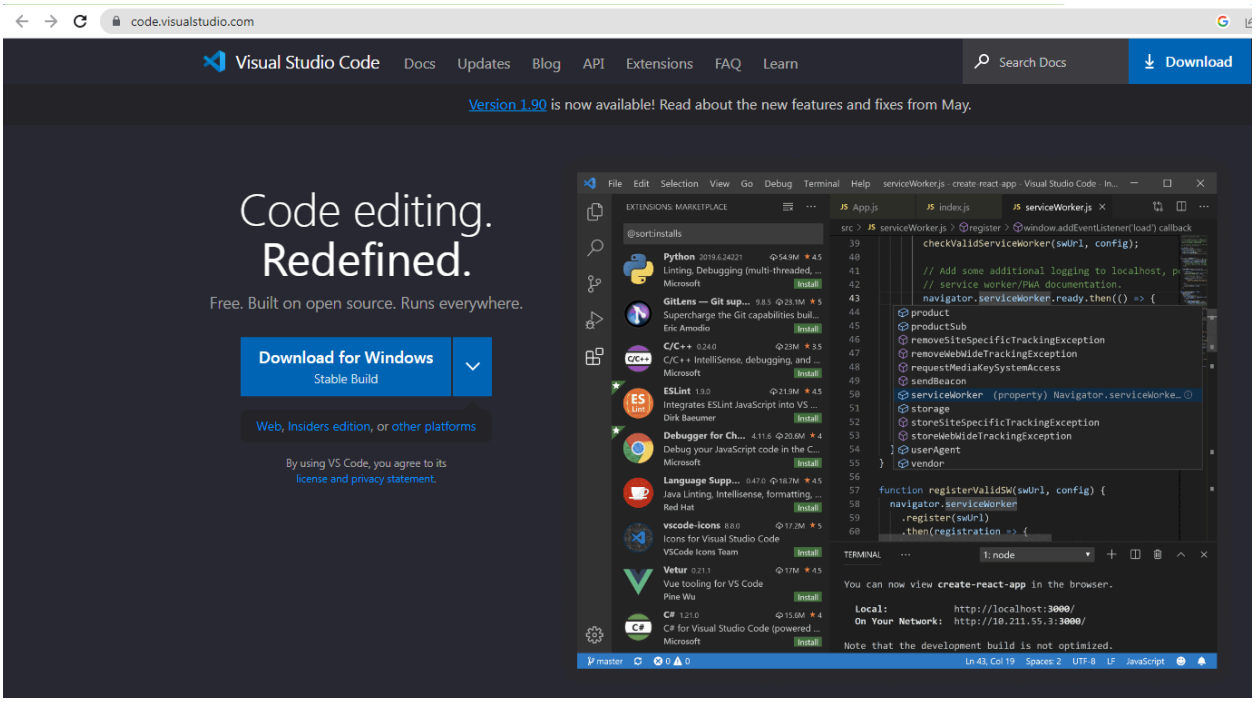

Рис. 2. Сторінка для скачування текстового редактору Microsoft Visual Studio Code

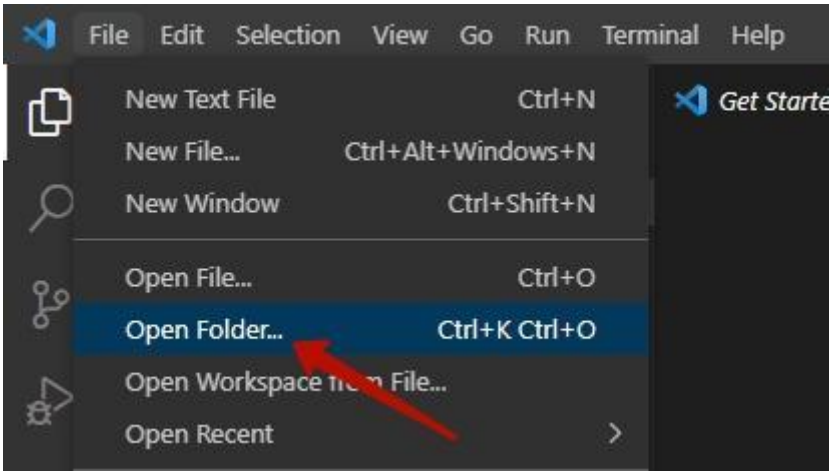

Рис 3. Відкриття робочої папки в Microsoft Visual Studio Code

Рис 4. Вибір робочої папки в Microsoft Visual Studio Code

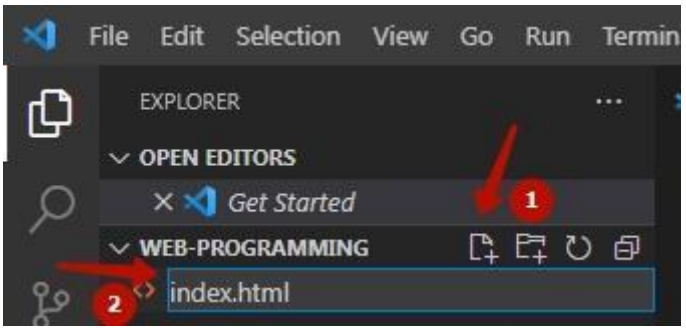

Рис 5. Створення файлу index.html в робочій папці в Microsoft Visual Studio Code

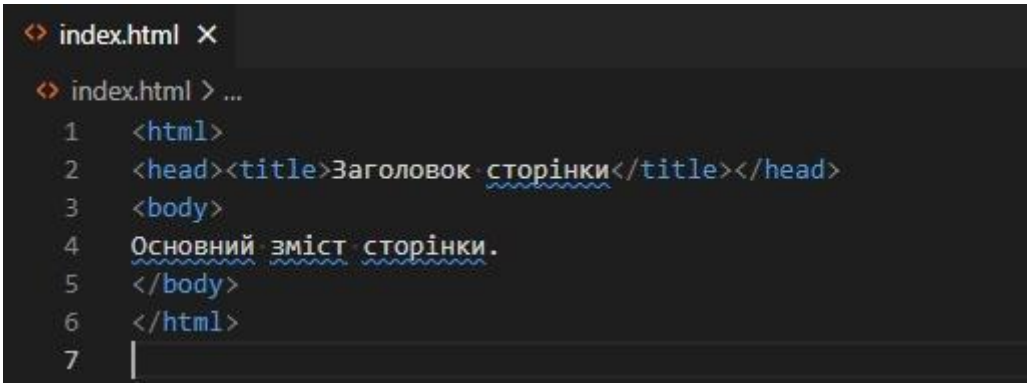

Рис. 6. Вставка коду в робоче вікно в Microsoft Visual Studio Code

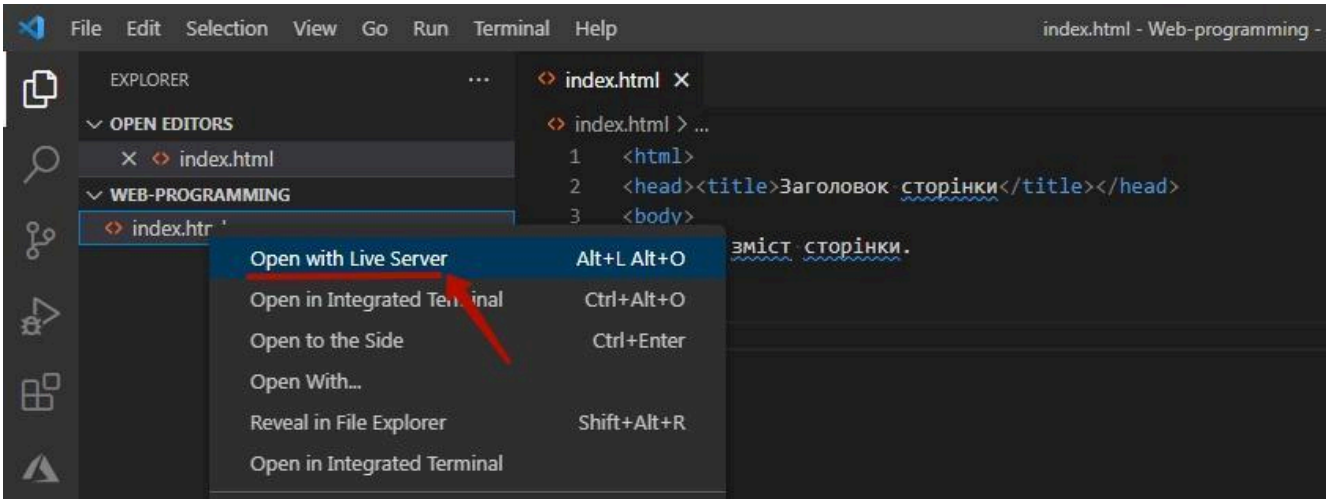

Рис. 7. Відкриття результату із використанням Live Server в Microsoft Visual Studio Code

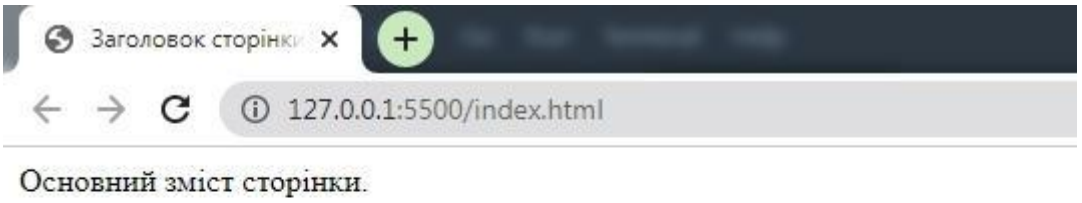

Рис. 8. Результуюча сторінка

Розділ 2. CSS. Каскадний аркуш стилів.

2.1. Історія створення CSS.

Первинно HTML був розроблений для верстки наукових текстів, проте Інтернет досить швидко з мережі, орієнтованої на науковий обмін переріс на загальну комунікаційну мережу. Відповідно недостатність можливостей HTML для задоволення всіх запитів, які з'явилися стала очевидною.

В грудні 1996 року була презентована технологія, яка дозволяла оформлювати сторінки із використанням кольорів, стилів текстів, відступів. Дана технологія мала назву CSS (Cascading Style Sheets), якщо дослівно – каскадні аркуші стилів.

Дана технологія, дозволяла визначати класи для елементів, а також призначати класам відповідне оформлення. При цьому CSS може підключатися до HTML в самому тегу, в тому самому документі та в зовнішньому документі.

2.2. Підключення CSS до HTML

Підключення CSS в тегу:

```
<p style = "color:red;">Текст відобразиться червним</p>
```

Підключення CSS в документі:

```
<html>
<head><title>Заголовок сторінки</title>
<style>
.p1{
  color:red;
}
</head>
<body>
<p class = "p1"> Текст відобразиться червним </p>
      </body>
</html>
```

Підключення CSS в зовнішньому документі:

Файл index.html	Файл style.css*,

<html>
<head><title>Заголовок сторінки</title>
<link rel="stylesheet" href="style.css"> </head>
<body>
 Текст відобразиться червним
</body>
</html>
	.p1{
	color:red;
	}

* - файл style.css знаходиться в тій самій папці, що і index.html

2.3. Структура CSS

Селектор у CSS є ключовим елементом, що використовується для визначення того, до яких HTML-елементів будуть застосовуватися стилі. Термін "селектор" походить від слова "select", що означає "вибирати". Селектор дозволяє вибрати один або декілька елементів на сторінці, яким буде призначено певне оформлення.

2.3.1. Селектор

Селектор - це набір команд, які визначають вибір певних елементів на веб-сторінці. Селектор може відповідати жодному елементу, одному елементу або всім елементам на сторінці.

2.3.2. Блок оголошень

Селектор відбирає елементи, і їм призначається певне оформлення. Це оформлення задається у блоці оголошень, який містить пари "властивість-значення".

2.3.3. CSS-правило

Разом селектор та блок оголошень утворюють CSS-правило. Весь CSS-файл складається з набору таких CSS-правил.

Рис. 9. Структура CSS правила

2.3.4. Підключення CSS до HTML

Розглянувши, як підключити CSS до HTML, тепер розглянемо, як змінювати оформлення HTML-елементів.

2.3.5. Зміна оформлення HTML-елементів без змін в HTML-коді

Для початку розглянемо, як можна змінити оформлення HTML-елементів без будь-яких змін у самому HTML-коді.

Це можливо, оскільки в CSS ми можемо перевизначати значення оформлення тегів за замовчуванням. Тобто ми можемо змінювати розмір, колір, оформлення, шрифт тощо для заголовків H1-H6, посилань, абзаців тощо.

2.3.6. Селектор типу.

За допомогою такого селектора ми зможемо змінити оформлення тегів за замовчуванням. Для цього в якості селектору нам треба вказати назву тегу

CSS

```
/* Змінюємо колір тексту для всіх заголовків H1 */
 h1 { color: blue; }
/* Встановлюємо інший розмір шрифту для всіх абзаців */
p { font-size: 18px; }
/* Змінюємо оформлення для всіх посилань */
a { text-decoration: none; color: red; }
```

HTML

```
<h1>Заголовок синього кольору</h1>
<p>Текст абзацу розміром 18 пікселів</p>
<a href = "">Текст посилання червоно кольору без підкреслювання</a>
```

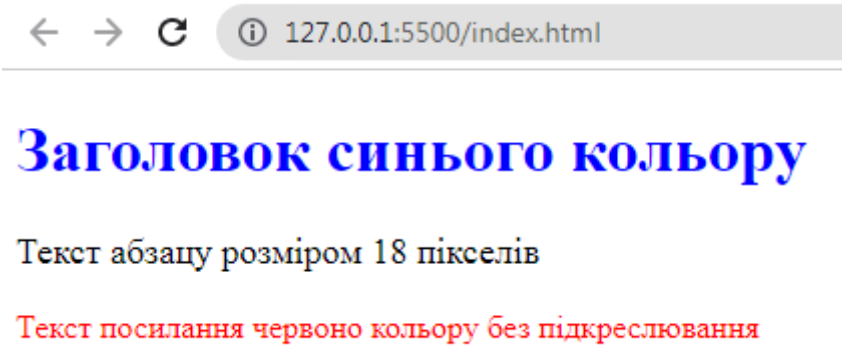

Рис. 10. Результат виконання коду

2.3.7. Селектор класу

Дозволяє змінити всі елементи, які мають відповідний клас. Для його визначення необхідно поставити крапку, а після неї написати назву класу.

CSS

```
.blue_color {
    color: blue;
}
.middle_size {
    font-size: 18px;
}
.link_decoration{
    text-decoration: none;
    color: red;
}
```

HTML

```
<h1 class = 'blue_color'>Заголовок синього кольору</h1>
<p class = 'middle_size'>Текст абзацу розміром 18 пікселів</p>
<a href = "" class = 'link_decoration '>Текст посилання червоно кольору без підкреслювання</a>
```

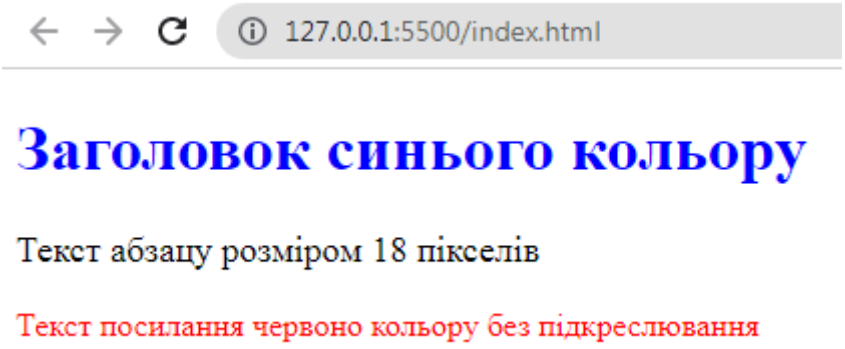

Рис. 11. Результат виконання коду

2.3.8. Селектор класу

Дозволяє змінити всі елементи, які мають відповідний ідентифікатор. Для його визначення необхідно поставити символ решітки, а після неї написати назву ідентифікатора.

CSS

```
#main_header {
    color: blue;
}
#base_paragraf {
    font-size: 18px;
}
#popular_link{
    text-decoration: none;
    color: red;
}
```

HTML

```
<h1 id = 'main_header'>Заголовок синього кольору</h1>
<p id = 'base_paragraf'>Текст абзацу розміром 18 пікселів</p>
<a href = "" id = 'popular_link'>Текст посилання червоного кольору без підкреслення</a>
```

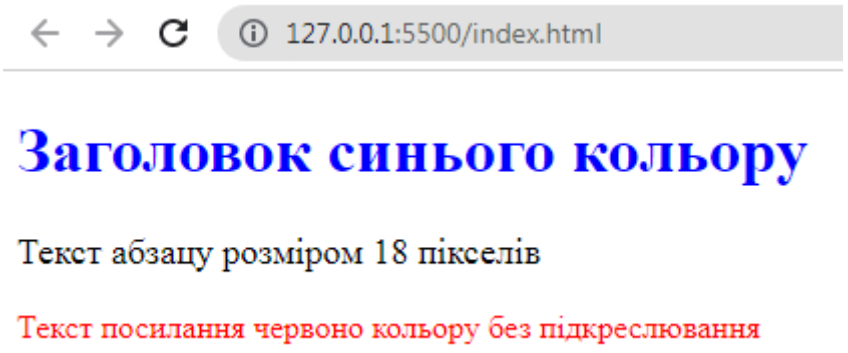

Рис. 12. Результат виконання коду

Розділ 3. Створення сучасної Web-сторінки

3.1 Семантичні теги HTML5

3.1.1. Загальний опис семантичних тегів

Спочатку єдиним тегом HTML, який відповідав прямокутному блоку на сайті, був . І з його допомогою проводилася вся верстка. Верхня частина сайтів дуже часто позначалася , основна частина сайту - , нижня частина сайту - .

При створенні нового стандарту (HTML5) були додані теги, які характеризують своє зміст. Тобто для верхньої частини сайту стали використовувати тег <header></header>, для основної - <main></main>, для нижньої частини сайтів - <footer></footer>.

3.1.2. Короткі описи семантичних тегів

- Header - вступна частина документа.
- Nav - основна навігація.
- Article - незалежна частина документа або сайту. Тобто якщо перемістити контент, який знаходиться в article, в інше місце, він не втратить своєї значущості.
- Section - загальний розділ документа (може бути частиною статті або іншого блоку).
- Main - основний зміст body - унікальний для сторінки, за винятком контенту, який може повторюватися на кількох сторінках, наприклад, навігація. Тобто те, що робить сторінку унікальною.
- Aside - блок для сайдбару зі змістом, відповідним сторінці, але таким, що має менше значення.

- Footer - нижня частина. Footer може задаватися як всьому документу, так і його окремим частинам.

3.1.3. Деякі правила семантичних тегів

- Header не можна поміщати всередину елементів: footer, address, header.
- Не всі групи посилань повинні бути обгорнуті у nav, цей елемент призначений для розділів, які складаються з основних навігаційних блоків.
- Section - не є блоком-обгорткою (для цього використовуємо div).
- Main - не може бути вкладеним в article, aside, footer, header або nav.

3.2. Потік документа

3.2.1 Логіка потоку в HTML

В HTML формування елементів на сторінці відбувається зверху вниз відповідно до схеми документа. Слой, розміщений у самому верху коду, відобразиться раніше шару, який розташований у коді нижче. Така логіка дозволяє легко прогнозувати результат виводу елементів і керувати ним.

3.2.2. Порядок виводу об'єктів

Порядок виводу об'єктів на сторінці називається «потоком». Це означає, що елементи відображаються на сторінці у тому ж порядку, в якому вони розміщені в HTML-коді.

3.3. Технологічний процес створення сайту

HTML та CSS дозволяють створювати структуру і оформлення веб-сторінок. Процес створення сайту часто включає перетворення дизайнерських макетів (картинок) у HTML-шаблони. Для ефективного освоєння цієї сфери корисно виділити кілька основних блоків:

3.3.1. Рядкові властивості

Рядкові властивості застосовуються до строчних (інлайнових) елементів, таких як `<span>`, `<a>`, `<strong>`, та інших. Вони змінюють оформлення цих елементів, не впливаючи на блокову структуру.

3.3.2. Блокові властивості

Блочні властивості застосовуються до блочних елементів, таких як `<div>`, `<section>`, `<article>`, та інших. Вони змінюють оформлення та розташування цих елементів на сторінці. Приклад блочних властивостей:

3.3.3. Псевдокласи та псевдоелементи

Псевдокласи та псевдоелементи дозволяють стилізувати елементи на основі їх стану або позиції в документі.

3.3.4. Позиціонування елементів

Позиціонування дозволяє точно розміщувати елементи на сторінці, використовуючи властивості `position`, `top`, `right`, `bottom`, та `left`.

3.3.5. Верстка сторінок

a. Горизонтальне та вертикальне вирівнювання елементів

Верстка сторінок включає різні техніки для горизонтального та вертикального вирівнювання елементів, а також використання сучасних технологій для розміщення елементів на сторінці.

b. Технології flexbox та grid

Flexbox та Grid є потужними інструментами для створення складних макетів.

3.4. Приклади сторінки.

3.4.1. Властивості рядкових елементів.

HTML

```
    <p class="p1">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p2">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p3">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p4">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p5">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p6">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
```

CSS

```
.p1 {
    font-family: Arial; /* родина шрифтів */
}

.p2 {
    font-size: 28px; /* розмір шрифта */
}

.p3 {
    font-style: italic; /* начертання (курсив) */
}

.p4 {
    font-weight: bold; /* насиченість (жирний шрифт) */
}

.p5 {
    font-variant: small-caps; /* малі заглавні букви (капітелі) */
}

.p6 {
    font: bold italic 26px Arial; /* вказання всіх показників у одному властивості */
}
```

3.4.2. Інші властивості рядкових елементів

color: green; /* колір шрифта */

text-decoration: underline; /* підкреслення */

background-color: red; /* колір фону */

Можна об'єднувати декілька властивостей шрифту в одному виразі.

font: bold 26px Verdana; /* жирний шрифт 26px Verdana */

3.4.3. Властивості тексту

```
     <p class="p1">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id
```

```
nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p2">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p3">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p4">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p5">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p6">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
     <p class="p7">Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio eum placeat eligendi ex tenetur omnis. Sapiente, eaque delectus. Ad voluptatibus eum id nobis maxime architecto pariatur, impedit dignissimos et sint!</p>
```

```
.p1 {
    letter-spacing: 5px; /* відступ між буквами */
}
.p2 {
    line-height: 30px; /* висота рядка */
}
.p3 {
    text-align: center; /* вирівнювання тексту всередині елементу */
}
.p4 {
    text-decoration: underline; /* підкреслення */
}
.p5 {
    text-indent: 50px; /* відступ від початку абзацу */
}
.p6 {
    text-transform: uppercase; /* всі великі літери */
}
.p7 {
    word-spacing: 20px; /* відступ між словами */
}
```

3.4.4. Блокові властивості

Блокові властивості - ті, що описують зовнішні та внутрішні відступи блокових HTML-елементів (зовнішній відступ, внутрішній відступ, межа).

```
<div class="d d1">Text Text Text Text Text Text Text Text Text </div>
<div class="d d2">Text Text Text Text Text Text Text Text Text </div>
<div class="d d3">Text Text Text Text Text Text Text Text Text </div>
<div class="d d4">Text Text Text Text Text Text Text Text Text </div>
<div class="d d5">Text Text Text Text Text Text Text Text Text </div>
```

```
.d {
    width: 100px; /* ширина блоку */
    height: 100px; /* висота блоку */
    background: brown; /* фон коричневий */
}

.d1 {
    margin: 10px; /* зовнішній відступ */
}

.d2 {
    padding: 10px; /* внутрішній відступ */
}
```

```
.d3 {
    border: 10px solid black; /* границя: ширина 10 пікселів, суцільна, чорна */
}

.d4 {
    margin-left: 50px; /* лівий зовнішній відступ */
}

.d5 {
     margin: 0 auto; /* зовнішній відступ: зверху і знизу - 0, зліва і справа -
автоматичний */
}
```

3.4.5. Інші блокові властивості

Інші блокові властивості:

margin-top:20px; - Зовнішній відступ (в даному випадку - зверху 20 пікселів). Варіанти margin-bottom (відступ знизу), margin-left, margin-right

padding-top:20px; - внутрішній відступ (у разі – зверху 20 пікселів). Варіанти padding-bottom (відступ знизу), padding-left, padding-right

Об'єднання для margin, padding:

margin:10px 15px 20px 25px; (top right bottom left)

Вказання декілька значень в одній властивості.

```
<div class="d d1">Text Text Text Text Text Text Text Text Text </div>
<div class="d d2">Text Text Text Text Text Text Text Text Text </div>
<div class="d d3">Text Text Text Text Text Text Text Text Text </div>
<div class="d d4">Text Text Text Text Text Text Text Text Text </div>
<div class="d d5">Text Text Text Text Text Text Text Text Text </div>
```

```
.d{
    width: 100px;
    height: 100px;
    background: brown;
    border-bottom: 1px dotted black;
}
.d1{
    padding:10px 20px 30px 40px;
}
.d2{
    padding: 10px 20px 30px;
}
.d3{
    padding:10px 20px;
}
.d4{
    border-left:10px solid black;
    border-right:10px dotted blue;
    border-top:10px  dashed yellow;
    border-bottom:10px double green;
}
.d5{
    border-left:10px groove #00f;
    border-right:10px ridge #00e;
    border-top:10px  inset #0f0;
    border-bottom:10px outset #00FE00;
}
```

Різні моделі розрахунку ширини блоків

```
<div class="d border-box">Lorem ipsum dolor </div>
```

```
<div class="d content-box">Lorem ipsum dolor </div>
```

```
.d{
    width: 100px;
    height: 100px;
    margin: 20px;
    padding: 20px;
    border: 10px solid #212121;
    background-color: #009688;
}
.border-box{
    box-sizing: border-box;
}
.content-box{
    box-sizing: content-box;
}
```

3.4.6. Псевдоелементи

Псевдоелементи дозволяють за допомогою CSS додавати елементи, яких немає у CSS. При цьому такі елементи розміщуватимуться в місці. пов'язаному з елементами, на які вказує селектор, а також їх положенням на сторінці.

Псевдоелементи: "до елемента", "після елемента", "перший символ", "перший рядок".

```
<div class="wrapper">
   <p class = "book"> Lorem ipsum dolor sit, amet consectetur adipisicing elit. Iste vitae temporibus velit consequuntur hic molestiae debitis ipsa alias nesciunt libero! </p>
    <p> Forem ipsum dolor sit, amet consectetur adipisicing elit. Iste vitae temporibus velit consequuntur hic molestiae debitis ipsa alias nesciunt libero! </p>
    <p> Lorem ipsum dolor sit, amet consectetur adipisicing elit. Iste vitae temporibus velit consequuntur hic molestiae debitis ipsa alias nesciunt libero! </p>
</div>
```

```
.book::first-line {
    color: #000;
}
.book::first-letter {
    font-family: 'Berkshire Swash', cursive;
    color: red;
    font-size: 25px;
}
.book::selection{
    background: violet;
    color:black;
}
.wrapper {
    width: 30%;
    margin: auto;
    padding: 20px;
    font-size: 20px;
    color: #fff;
    background: #070;
  }
```

```
<p class = "qt">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Architecto, alias quasi quae eum quibusdam rerum ad corporis, sequi vero, laboriosam consectetur illum suscipit. Optio, id totam consequuntur accusamus amet quidem?</p>
```

```
.qt::before{
```

```
    content:'"';
}
.qt::after{
    content:'"';
}
```

3.4.7. Псевдокласи.

Псевдокласи дозволяють змінювати оформлення залежно від певних станів елемента (наведення миші, фокус елемента, клік миші), а також для посилань (властивості посилання, властивості переглянутих посилання).

```
<div class="d">Text</div>
```

```
.d{
    height: 100px;
    width: 100px;
    border-radius:10px;
    border:1px solid;
    background: brown;
}
.d:hover{
    border:5px solid;
}
```

```
<input type="text">
<input type="text">
<input type="text">
<input type="text">
```

```
input:focus{
       background: #ddd;
}
```

3.4.8. Колір

Колір в CSS може задаватися трьома двозначними шістнадцятковими цифрами, трьома двоцифровими шістнадцятковими цифрами, показниками rgb (3 цифри від 0 до 255 або від 0% до 100%), показниками rgba (те ж що і rgb + значення альфа (прозорість) від 0 до 1).

```
<div class="d simple"></div>
<div class = "d alfa"></div>
<div class = "d d2">Text<div class="d alfa2 d3"></div></div>
```

```
.d{
    width:100px;
    height:100px;
}
.simple{
    background: red;
}
.alfa {
    background: linear-gradient(
```

```
    to right,
    rgb(0%, 0%, 0%, 0),
    rgb(100%, 0%, 0%)
    );
}
.alfa2 {
    background:  rgba(255, 10, 0, 0.6);
}
.d2{
    position: absolute;
}
.d3{
    position:relative;
    top:-20px;
}
```

3.4.9. Фон

Фон CSS може бути однотонним (один колір), градієнтним (перехід від одного відтінку до іншого), а також може заповнюватися зображенням.

```
background-color: #ccf;
background-image:
url(https://images.pexels.com/photos/724712/pexels-photo-724712.jpeg?w=630&h=500&
auto=compress&cs=tinysrgb);
background-repeat: no-repeat;
background-position: center center;
background-attachment: fixed;
background-size: contain;
```

3.4.10. Типи відображення елементів (властивість display)

Використовуючи значення властивості display можна будь-якому елементу задати відображення як рядковому (display:inline;) - вбудовується в рядок не має ширини та висоти, блоковому (display:block;) - займає все доступне місце та рядково-блочному (display:inline-block;) - вбудовується в рядок, але має свої ширину та висоту.

```
      <p>Lorem ipsum dolor sit amet, <span class = "inline">inline inline inline
inline</span>. Vivamus sagittis augue id sagittis vestibulum. Nam sodales enim a
dolor rhoncus, ut porta eros varius. Ut ultricies finibus felis quis auctor.
Proin ultricies massa ac magna mollis, et venenatis mauris elementum. Sed
consequat sed nisi id placerat. Morbi eu eros sit amet metus scelerisque
efficitur. Proin vestibulum, est eget consectetur porttitor, justo nisl mollis
orci, eget consequat leo orci sit amet lacus.</p>
      <p>Lorem ipsum dolor sit amet, <span class = "block">block block block
block </span>. Vivamus sagittis augue id sagittis vestibulum. Nam sodales enim a
dolor rhoncus, ut porta eros varius. Ut ultricies finibus felis quis auctor.
Proin ultricies massa ac magna mollis, et venenatis mauris elementum. Sed
consequat sed nisi id placerat. Morbi eu eros sit amet metus scelerisque
efficitur. Proin vestibulum, est eget consectetur porttitor, justo nisl mollis
orci, eget consequat leo orci sit amet lacus.</p>
      <p>Lorem ipsum dolor sit amet, <span class = "inline-block">inline-block
inline-block inline-block</span>. Vivamus sagittis augue id sagittis vestibulum.
Nam sodales enim a dolor rhoncus, ut porta eros varius. Ut ultricies finibus
felis quis auctor. Proin ultricies massa ac magna mollis, et venenatis mauris
elementum. Sed consequat sed nisi id placerat. Morbi eu eros sit amet metus
scelerisque efficitur. Proin vestibulum, est eget consectetur porttitor, justo
nisl mollis orci, eget consequat leo orci sit amet lacus.</p>
```

```
span{
  border:1px solid;
  height:30px
}
.inline{
  display:inline;
  color:green;
  width:300px;
}
.block{
  display:block;
  color:blue;
}
.inline-block{
  display:inline-block;
  color:red;
  width:300px;
}
```

3.4.11. Псевдокласи. hover, active

У CSS закладено механізм зміни оформлення елемента при настанні певної події миші (наведення на елемент – псевдо клас hover; клік на елементі – псевдоклас active)

```
<div class = "block">!Lorem ipsum dolor sit amet consectetur adipisicing elit. Dignissimos unde quidem debitis commodi ipsum minima, consequatur, aspernatur consectetur similique corrupti praesentium, corporis at reprehenderit laboriosam error perferendis magnam quia obcaecati.</div>
```

```
.block{
    width:200px;
    height:200px;
    border:10px solid black;
}
.block:hover{
    border:10px solid red;
    cursor: pointer;
}
.block:active{
    border:10px solid blue;
}
```

3.4.12. Медіазапити

На сьогодні кількість пристроїв, за допомогою яких користувачі переглядають сайти дуже велика. При цьому існує велика кількість самих пристроїв для перегляду сайтів: мобільні телефони, планшети, екрани персональних комп'ютерів.

Логічно, що сайт на мобільному телефоні відрізнятиметься від сайту на планшеті та від сайту, який ми переглядатимемо на екрані.

Для реалізації можливість змінювати відображення різних елементів при різному розмірі екрана, на якому відбувається відображення сайту, використовується технологія медіазапитів (media queries)

Ця технологія реалізується в CSS

```
@media (min-width: 1200px) {
    article {
width: 800px;
    }
}
```

Дана форма запису говорить про те, що якщо ширина екрану поточного пристрою для відображення сайту всі елементи article будуть мати ширину 800 пікселів.

Демонстраційний приклад:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Домашнее задание</title>
    <link rel="stylesheet" href="css/all.css">
</head>
<body>
    <article>
<div id="text">
    <div id="hello">Hello!</div>
</div>
    </article>
</body>
</html>
```

```
body {    background: #EE66AA;}
article {    border: 1px solid black;}
#text {    font-size: 48px;}
#hello {    display: none;}
article {    width: 300px;    height: 200px;    background: yellow;}
#text:after {    content: "320";}

@media (min-width: 480px) {
    article {  width: 470px;  height: 300px;    background: aqua;  }
    #text:after {  content: "480";  }
}
@media (min-width: 768px) {
    article {    width: 768px;   height: 400px;   background: green;  }
    #text:after {     content: "768";   }
}
@media (min-width: 960px) {
    article {   width: 960px;    height: 500px;     background: blue;    }
    #text:after {    content: "960";  }
}
@media (min-width: 1200px) {
    article {     width: 1200px;     height: 500px;     background: gray;    }
    #text:after {   content: "1200";   }
}
```

Розділ 4. FlexBox

На сьогоднішній день основною технологією, яка використовується для верстки сайтів, є технологія FlexBox. Вона передбачає використання властивості display із значенням flex (display:flex;)

Існує велика кількість статей, що пояснюють принципи роботи цієї технології верстки.
https://tproger.ru/translations/how-css-flexbox-works/
https://developer.mozilla.org/ru/docs/Learn/CSS/CSS_layout/Flexbox

Технологія FlexBox розроблена на основі вже існуючого багаторічного досвіду професійної спільноти щодо створення сайтів. Ця технологія дуже добре описує всі основні ситуації, що зустрічаються у повсякденному процесі верстки.

Основні прийоми верстки за технологією FlexBox:

- Поставити блоки поруч один з одним - батькові поставити властивість display:flex;

- Поставити блоки поруч один з одним у зворотному порядку: додати батькові властивість flex-direction: row-reverse;
- Поставити блоки один під одним, але у зворотному порядку - додати батькові властивість row-column;
- Вирівняти блоки по центру - додати батькові властивість justify-content:center;
- Вирівняти блоки з правого краю (додати властивість батькові justify-content:flex-end;) варто зауважити, що ця властивість вирівнюватиме блоки з правого краю при значенні flex-direction: row; (значення за замовчуванням), а якщо буде вказано flex-direction: row-reverse; - то вирівнювання відбуватиметься лівим краєм, тобто. justify-content:flex-end; вирівнює текст до кінця напряму основної осі.
- Вирівняти блоки з лівого краю justify-content:flex-start; (За умови значення flex-direction: row;)
- Помірно розподілити блоки по всьому доступному місцю батька:
- justify-content:space-between; - крайні блоки будуть притиснуті до країв батька, відступи між блоками будуть рівномірними
- justify-content:space-around; - все вільне місце між блоками буде рівномірно розподілено навколо блоків. Тобто. якщо у нас n блоків, то вся вільна ширина w буде рівномірно розподілена навколо кожного блоку (1 вільний простір шириною w/n зліва і 1 1 вільний простір шириною w/n справа)
- justify-content:space-evenly; весь вільний простір буде розподілено рівномірно у проміжках від країв до олоків та між блоками

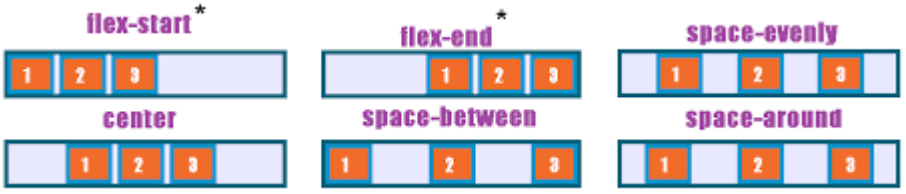

Рис. 13. Схема використання властивості justify-content

Частина 2. Програмування на JavaScript

2.1. Перші програми на JavaScript

2.1.1. Загальний опис мови програмування JavaScript

Для створення статичних сторінок використовується HTML+CSS. Проте сайти без алгоритмічної складової не задовольняють всіх потреб користувачів. Саме тому був сформований запит на мову програмування, яка може виконуватися в браузері і в 1995 році такою мовою став саме JavaScript.

JavaScript надав сторінкам динамічних властивостей: зміну елементів залежно від поведінки користувача на сторінці, включно зі зміною вмісту без перезавантаження сторінки, реалізація математичних обчислень та алгоритмів тощо. По сьогодні (тобто вже майже 30 років) JavaScript є основною та єдиною мовою програмування, яка виконується в браузері.

Спеціалістів, які володіють HTML, CSS та JavaScript називають FrontEnd-розробниками, тому що ці мови є вичерпним переліком мов розмітки та програмування, що можуть бути виконані в браузері (у лицьовій, клієнтській, тобто FrontEnd-частині сайту).

JavaScript - це інтерпретований мова програмування, яка виконується в браузері. Інтерпретована мова програмування означає те, що він виконується «на льоту». Код JavaScript підключається до HTML в самих файлах або може підключатися окремими файлами, принцип підключення чимось схожий на підключення CSS.

2.1.2 Підключення JavaScript до HTML

Вставка коду JavaScript в код HTML здійснюється кількома способами:

1. Вставка коду між тегами <script></script> у коді HTML+CSS
2. Підключення файлу із кодом, що знаходиться на тому ж комп'ютері, що і HTML-файл.

1. <script src="/path/to/script.js"></script>
2. Підключення файлу, що знаходяться в Інтернеті
3. <script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.3.0/lodash.js">
4. </script>

2.1.3. Виведення даних

Традиційне вивчення всіх мов програмування починається з виведення тексту Hello, World! на екрані (ця традиція була почата в 1978 р. у книзі Денніса Рітчі, розробника мови Сі та Браяна Карнігана).

Для виведення тексту в JavaScript використовується кілька можливостей:

a. **alert("Hello, World!");** - виведення тексту в службовому окні.
b. **console.log("Привіт, світ!");** - виведення тексту в консоли браузера.
c. в. **document.write("Привіт, світ!");** - виведення поточного HTML+CSS документа на екран браузера
d. Створює блок і вставляє в нього текст

```
var body = document.body;
var div = document.createElement('div');
div.innerHTML = 'Привіт, світ!';
body.appendChild(div);
```

2.1.4. Виведення тексту через службове вікно

Приклад програмного коду:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <script>
    alert("Hello World!");
  </script>
</body>
</html>
```

Результат виконання коду:

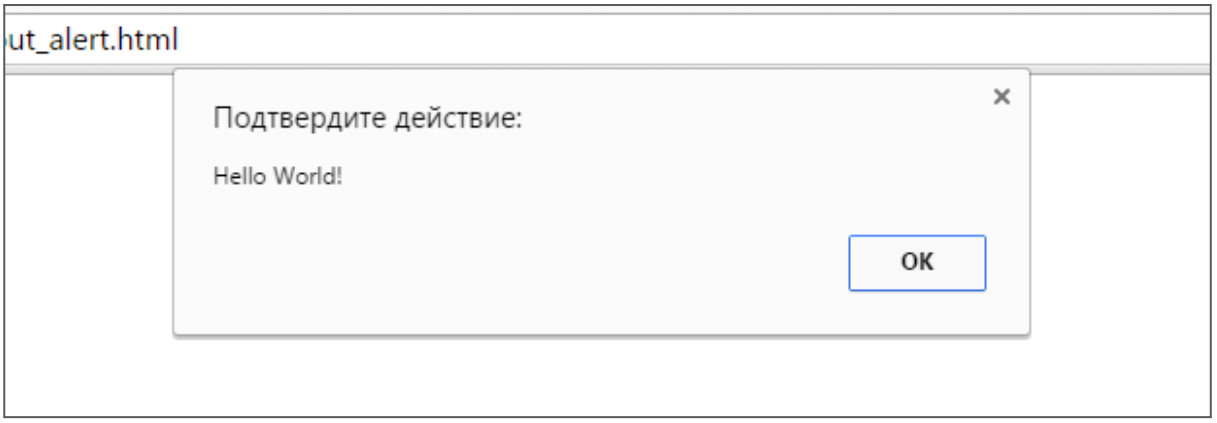

Рис. 14. Результат виконання команди alert

2.1.5. Виведення тексту до консолі:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <script>
   console.log("Hello World!");
  </script>
</body>
</html>
```

Нажимаем правую клавішу. Вибираємо "Переглянути код"

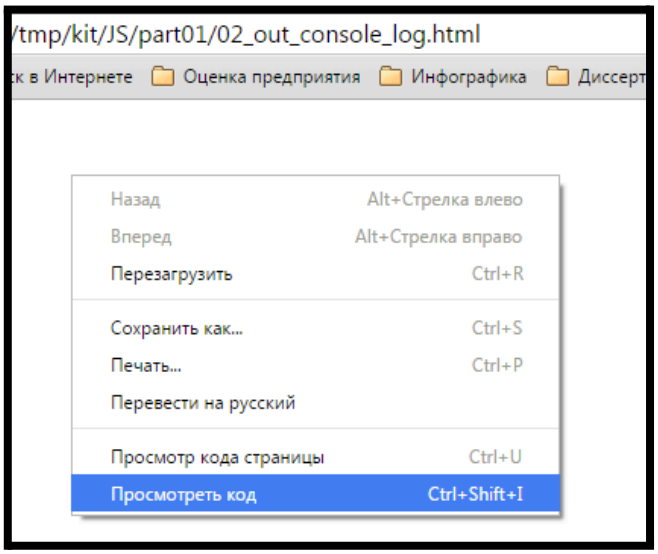

Рис. 15. Службове меню

У відкритому вікні вибираємо блок "console"

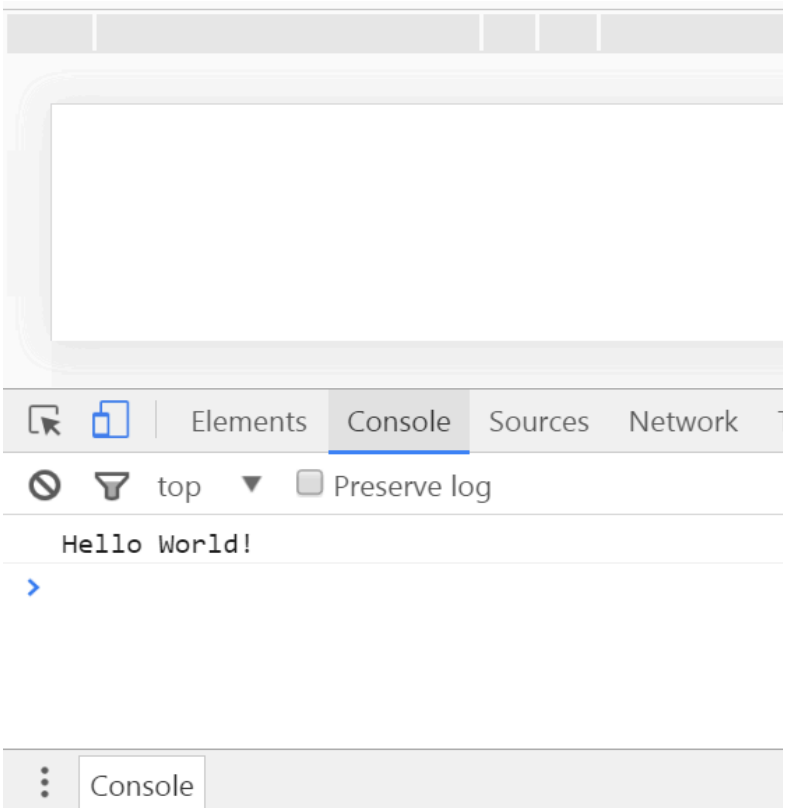

Рис. 16. Консоль браузера

2.1.6. Виведення тексту на екран:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <script>
     document.write("Hello World!");
  </script>
</body>
</html>
```

Hello World!

Рис. 17. Результат виконання коду document.write("Hello World!");

2.1.7. Динамічне додавання елемента на екран:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document. AppendChild</title>
</head>
<body>
```

```
  <script>
       var body = document.body;
       var div = document.createElement('div');
       div.innerHTML = 'Hello World!';
       body.appendChild(div);
  </script>
</body>
</html>
```

Результат роботи:

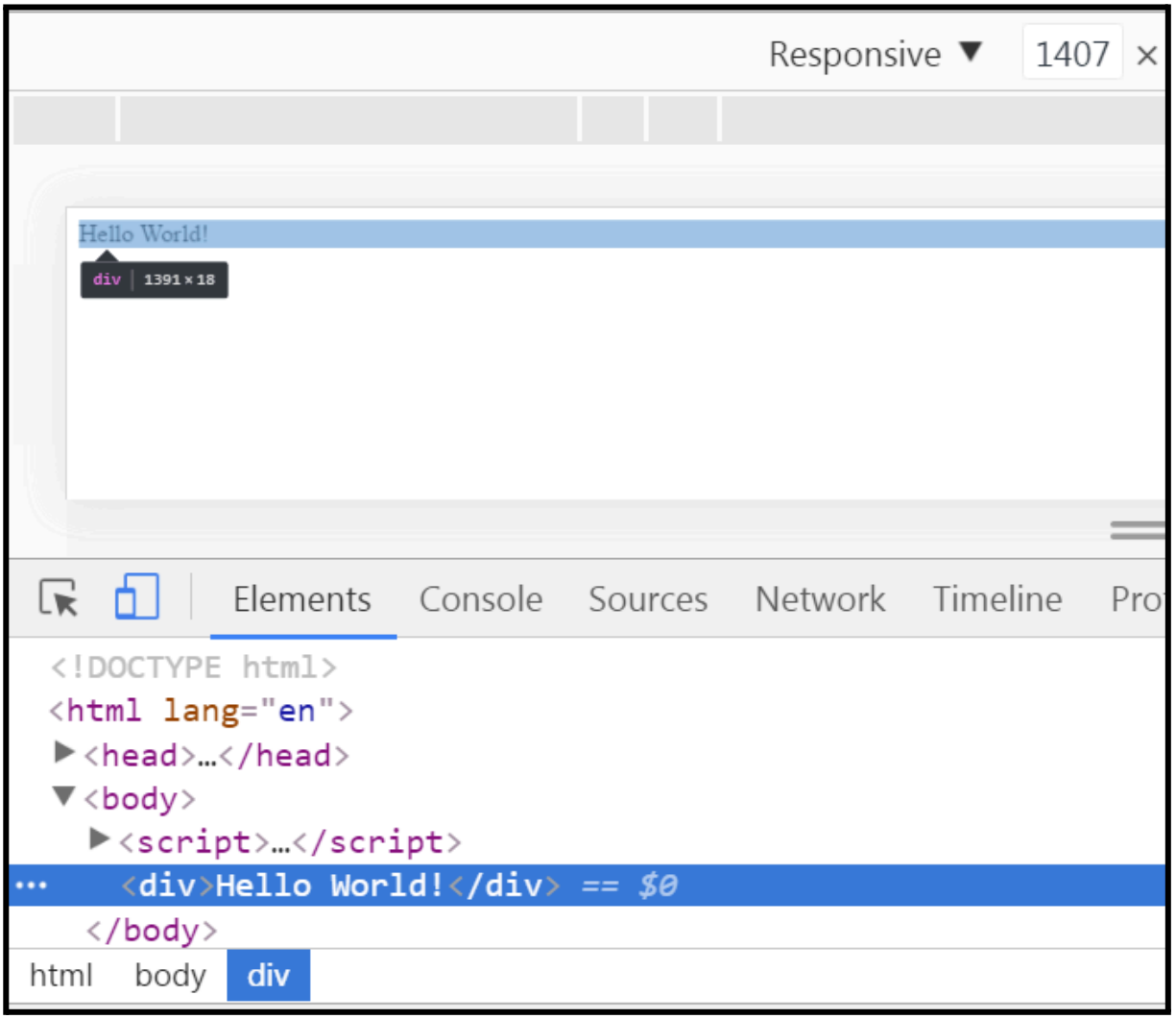

Рис. 17. Результат динамічного додавання елементу на екран;

Розділ 2. Змінні в JavaScript

2.1.1. Опис роботи зі змінними

1. Виведення тексту та обчислення - це початкова можливість, яка зазвичай не використовується окремо.
2. Особливість мов програмування в тому, що вони працюють із свою пам'яті. Роботу із пам'яттю - реалізується через змінні.
3. Змінні - це іменовані комірки пам'яті, в яких зберігається інформація.
4. Для виділення пам'яті і подальшої роботи з нею - ми визначаємо ім'я для такої комірки і надаємо їй значення: var a = 1;

5. Далі можна виконати операцію з назвою змінної так, якщо ми працювали з її значенням a = a+2; alert(a);

2.1.2. Приклад роботи зі змінними

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Vars. Main principes.</title>
</head>
<body>
  <script>
       var a=1; //Загадайте число
       a=a+2; //Добавьте к загаданному числу 2.
       alert(a);//"скажите" результат
  </script>
</body>
</html>
```

2.1.3. Введення даних

1. Програми пишуться для користувачів, вони мають взаємодіяти з користувачем.

2. Виводити на екран дані та проводити обчислення ми вже навчилися. Тепер потрібно навчитися вводити дані.

3. Введення даних у JavaScript здійснюється:

a. var a = prompt (); -введення даних з сервісного вікна

b. var a=Math.random(); - введення випадкового числа

c. <input id = "in1" value = "1"> <script>
  var in1 = document.getElementById("in1");
console.log(in1.value);
</script>

Читання даних з об'єктів на сторінки

2.1.4. Введення даних із використання діалогового вікна

```
<script>
       var a=prompt(); //Ввод через диалоговое окно
       alert(a+2); // Output as string
       a=parseInt(a); //преобразование строки в число
       alert(a+2);//Output as number
  </script>
```

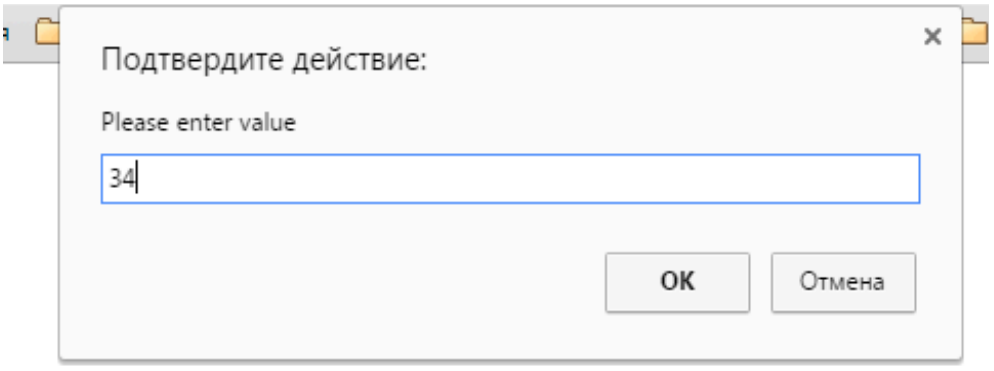

Рис. 20. Службове вікно для введення даних.

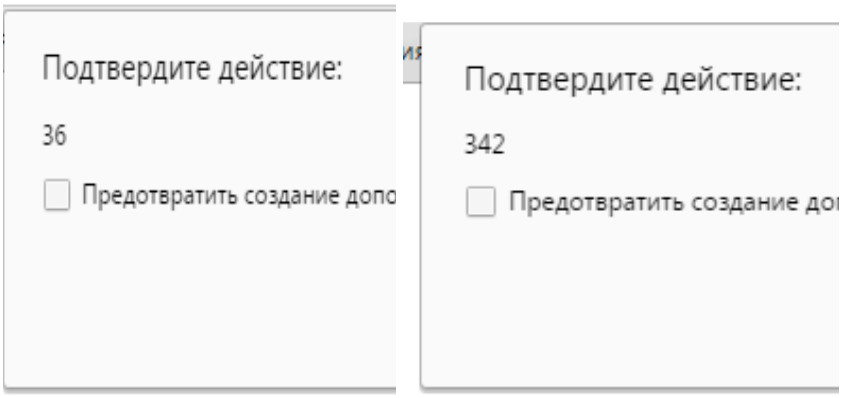

Рис. 21. Службове вікно для введення даних.

```
<script>
  var a=Math.random();
  console.log(a);
  a = Math.floor(a*10);
  console.log(a);
</script>
```

При кожному оновленні сторінки цифра змінюватиметься

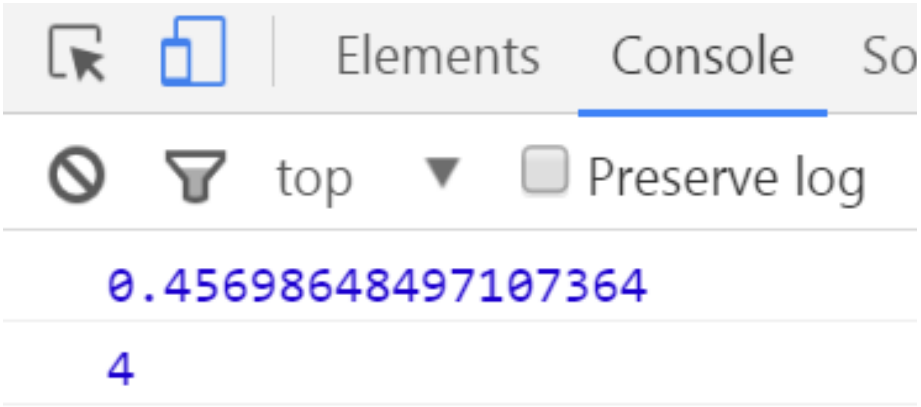

Рис. 22. Результат генерації випадкового числа

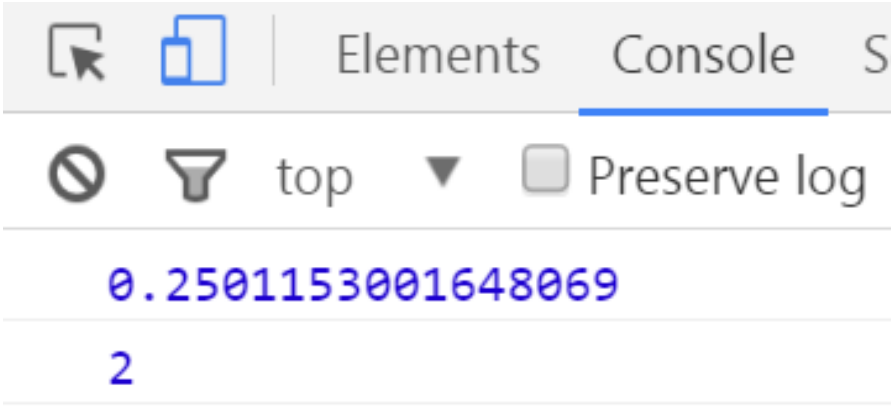

Рис. 23. Результат генерації випадкового числа

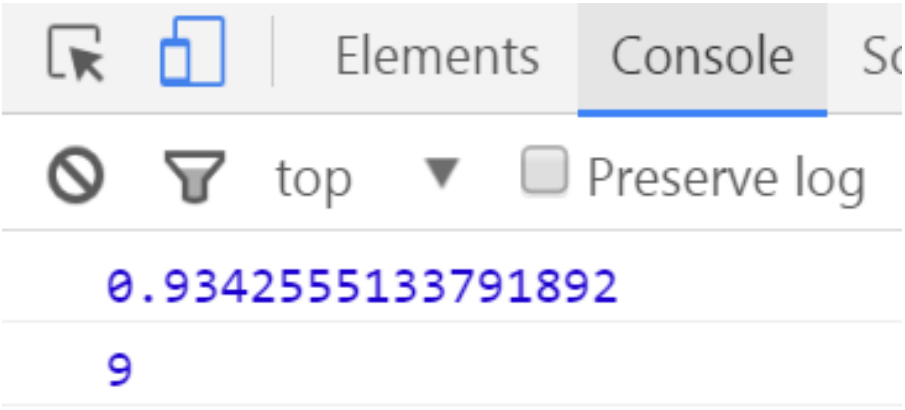

Рис. 24. Результат генерації випадкового числа

2.1.5. Виведення до консолі значення елементу input за замовчуванням

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Vars. Main principles.</title>
</head>
<body>
  <input id ="in1" value = "1">
  <script>
    var in1 = document.getElementById("in1");
    console.log(in1.value);
  </script>
</body>
</html>
```

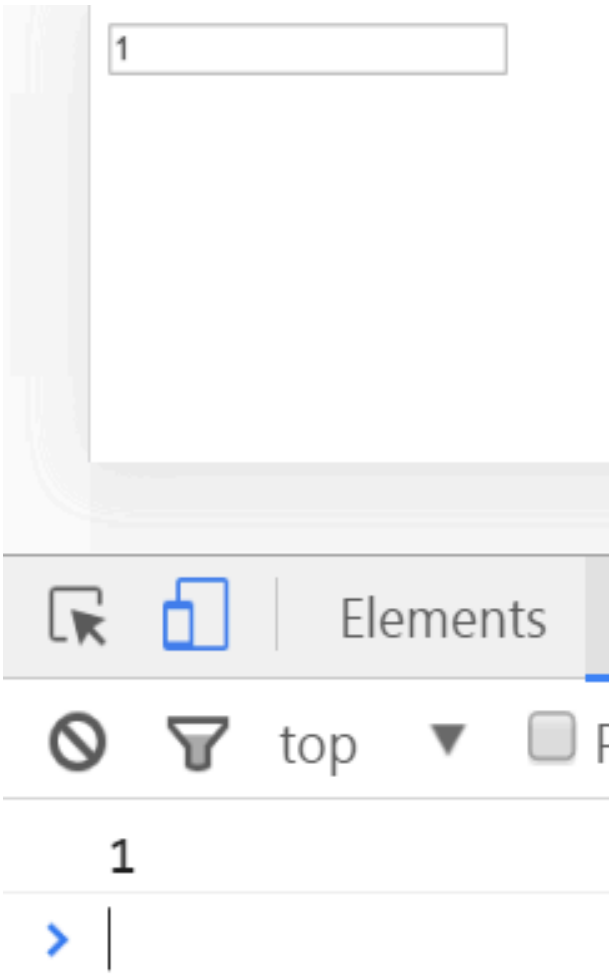

Рис. 25. Результат виведення до консолі значення елементу input за замовчуванням

2.1.6. Введення даних із подією

```
<input id ="in1" value = "1">
<button onclick = "f_out()">click</button>
<script>
  function f_out (){
    var in1 = document.getElementById("in1");
    console.log(in1.value);
  }
</script>
```

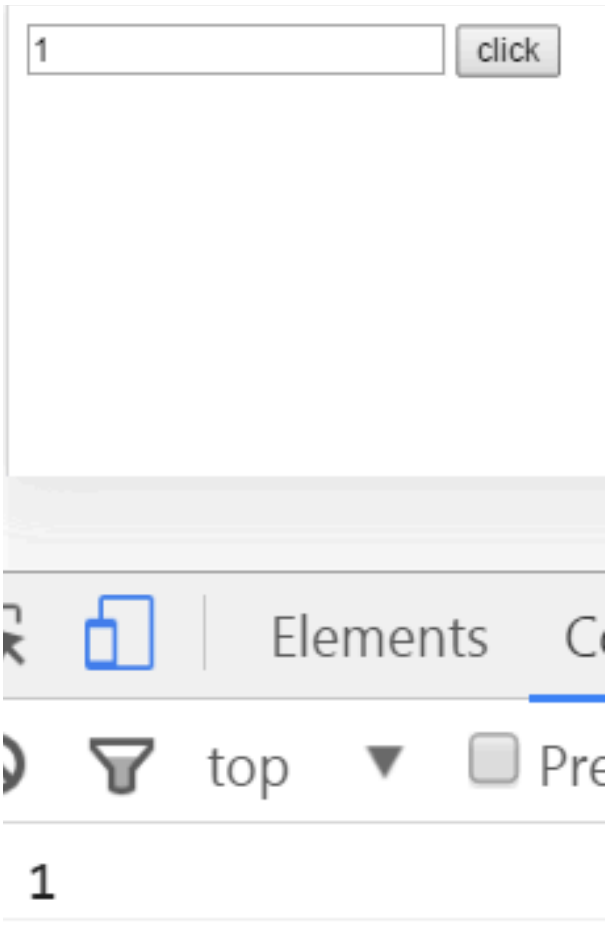

Рис. 25. Результат виведення до консолі значення елементу input за події (натискання кнопки)

Обробка введення значень може здійснюватися як при завантаженні сторінки, так і за певної події (натискання кнопки, наведення на елемент, ін.)

2.1.7. Типи даних

Для будь-якої мови типи значень, які вона може утворювати, визначаються типами змінних, які визначені для відповідної мови.

Для JavaScript це:

- **String**: представляє рядок
- **Number**: представляє числове значення
- **Boolean**: представляє логічне значення true або false
- **undefined**: вказує, що значення не встановлено
- **null**: вказує на невизначене значення

1. У людини є образна пам'ять, є логічна - у мовах програмування також виділяють різні типи даних, що зберігаються.
2. У JavaScript - 5 примітивних типів даних (числа, рядки, булев тип, тип "невизначене значення" і тип "Null"), а також тип "Об'єкти".
3. Щоб дізнатися тип змінної необхідно перед змінною вказати тип
4. Наприклад: console.log (typeof a);

```
var integer=5; //Целое число. Тип «number»
var fl=0.123; //Вещественное число.Тип «number»
var ch='c'; //Символ. Тип  «string»
var str="This is string" ; //Строка. Тип  «string»
var bool=true ; //Булев тип. Тип  «boolean»

console.log (typeof integer);
console.log (typeof fl);
console.log (typeof ch);
console.log (typeof str);
console.log (typeof bool);
```

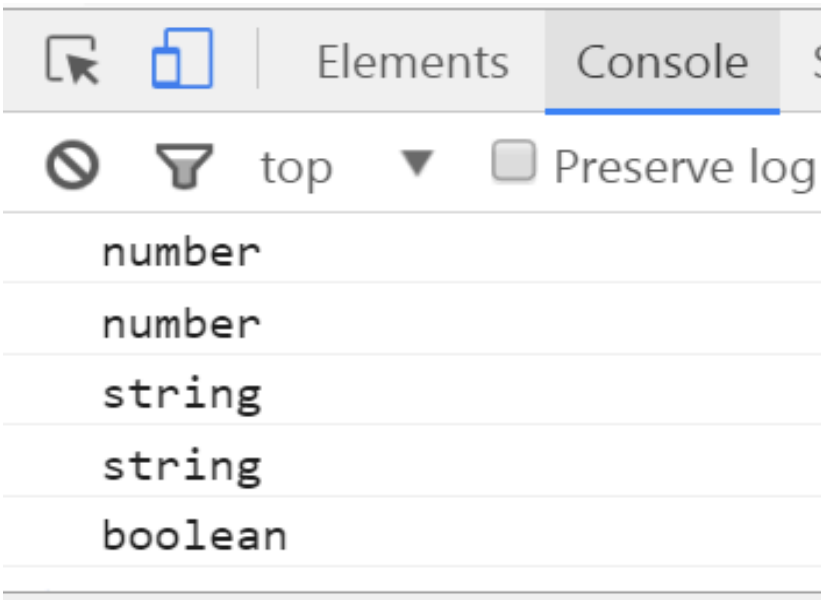

Рис. 18. Виведення до консолі типів змінних

2.1.8. Основні арифметичні дії

1. Однією з перших завдань, яку вирішували з допомогою мов програмування були математичні обчислення. І сьогодні обчислення є дуже важливою складовою будь-якої програми.
2. У JavaScript реалізовані основні арифметичні дії: "+", "-", "*","/"
3. Операція "залишок від розподілу" реалізована за допомогою оператора "%"
4. Операції "розподіл на ціло" на пряму JavaScript не реалізована, тому вона реалізується за допомогою: Math.floor(a/10);

```
<script>
  var a=5;
  var b=2;
  var c=a+b; console.log ("5+2 = ",c);
  c=a*b;      console.log ("5*2 = ",c);
  c=a-b;      console.log ("5-2 = ",c);
  c=a/b;      console.log ("5/2 = ",c);
  c=a%b;
  console.log ("5%2 = ",c,"(остаток от деления)");
  c=Math.floor(a/b);
    console.log  ("Math.floor(5/2)  =  ",c,"(деление  на  цело,  целая  часть  от
деления)");
  </script>
```

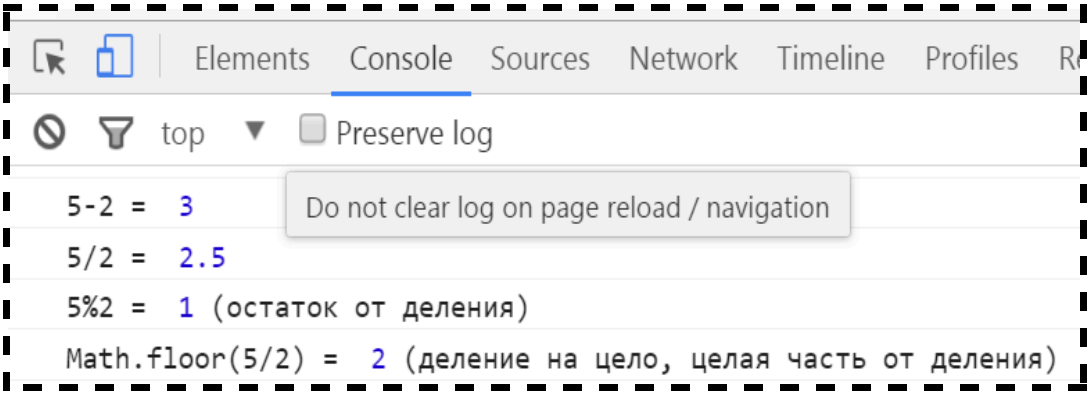

Рис. 19. Результат виконання коду із найпростішими арифметичними діями

2.1.9. Числові дані

Числа JavaScript можуть мати дві форми:

Цілі числа, наприклад, 35. Ми можемо використовувати як позитивні, і негативні числа. Діапазон використовуваних чисел: від -2^{53} до 2^{53}

Дробові числа (числа з плаваючою точкою), наприклад, 3.5575. Знову ж таки можна використовувати як позитивні, так і негативні числа. Для чисел із плаваючою точкою використовується той самий діапазон: від -2^{53} до 2^{53}

Наприклад:

```
let x = 45;
let y = 23.897;
```

Як роздільник між цілою та дробовою частинами, як і в інших мовах програмування, використовується точка.

Рядки

Тип string представляє рядки, тобто такі дані, які укладені в лапки. Наприклад, "Привіт світ". Причому ми можемо використовувати як подвійні, так і одинарні лапки: "Привіт світ" та "Привіт світ". Єдине обмеження: тип лапки, що закриває, повинен бути той же, що і тип відкриває, тобто або обидві подвійні, або обидві одинарні.

Якщо всередині рядка трапляються лапки, то ми їх повинні екранувати слішем. Наприклад, нехай у нас є текст "ТОВ "Спецвузавтоматика"". Тепер екрануємо лапки:

```
let itCompany = "ООО \"Спецвузавтоматика\"";
```

Також ми можемо всередині стоки використовувати інший тип лапок:

```
let itCompany = "ООО 'Спецвузавтоматика'";
let itCompany = ООО"Спецвузавтоматика"';
```

Тип Boolean

Тип Boolean являє собою булеві або логічні значення true і false (тобто так чи ні):

```
let isAlive = true;
let isDead = false;
```

null і undefined

Нерідко виникає плутанина між null і undefined. Отже, коли ми тільки визначаємо змінну без надання їй початкового значення, вона представляє тип undefined:

```
var isAlive;
console.log(isAlive); // выведет undefined
```

Надання значення null означає, що змінна має деяке невизначене значення (не число, не рядок, не логічне значення), але все-таки має значення (undefined означає, що змінна не має значення):

```
var isAlive;
console.log(isAlive); // undefined
isAlive = null;
console.log(isAlive); // null
isAlive = undefined;  // снова установим тип undefined
console.log(isAlive); // undefined
```

Розділ 3. Оператори вибору

3.1. Загальний опис оператору вибору.

Повторення: у минулих уроках ми навчилися з Вами отримувати дані від користувача (через службову команду prompt, через отримання значень у блоках input), проводити обчислення, використовувати змінні та виводити отриманий результат (через службове вікно командою alert, у консоль, через динамічну зміну елементів у браузері).

В даному розділі ми розберемо таку структуру JavaScript як розгалуження (оператор вибору). Ця конструкція використовується для моделювання бінарного рішення (якщо умова правильна - робимо дію 1, інакше - робимо дію 2).

Аналогічні операції робить і людина, коли перевіряє стан певного параметра.

Для прикладу візьмемо ситуацію, коли ми заходимо в кімнату і дивимося, якщо в кімнаті темно ми включаємо світло, якщо в кімнаті світло нічого не робимо.

Візьмемо як значення параметра, яке ми перевірятимемо відповідь на запитання "У кімнаті темно?" - якщо відповідь на це питання ТАК - включаємо світло, якщо НІ - то нічого не робимо.

У нашому повсякденному житті ми найчастіше собі ставимо якісні питання - "Чи світло в кімнаті?", "Чи тепло на вулиці?", "Чи достатньо у мене грошей?".

Так як людина - це складна обчислювальна машина вона може працювати навіть з такими нечіткими поняттями. Програма, як правило, відповісти на ці питання може тільки тоді, чи будуть задані чіткі критерії - стан "світло" - відповідає рівню освітленості у скільки одиниць і який зараз рівень освітленості (у скільки одиниць). Тобто. комп'ютер може порівнювати лише певні значення.

3.2 Форми оператору вибору

У JavaScript оператор вибору записується так:

скорочений запис (тільки справжня гілка)

```
if (a>0){
  console.log("Число а - додатне");
```

```
}
```

Повний запис (справжня та помилкова гілки)

```
if (a>0){
  console.log("Число а - додатне");
} else {
  console.log("Число а – від'ємне або дорівнює 0");
```

Розгалуження також можуть йти в комбінації один з одним:

```
if (a>0){
  console.log("Число а - додатне");
} else if (a ==0){
  console.log("Число а дорівнює 0");
} else {
  console.log("Число а – від'ємне 0");
}
```

Як оператори, за допомогою яких створюється умова, використовуються оператори порівняння:

- == дорівнює
- === тотожно дорівнює (порівнюються не тільки значення, а й типи змінних)
- != не дорівнює
  - □ більше
- < менше
- >= більше або дорівнює
- <= менше або дорівнює

Також розгалуження можуть включати складові умови. Для створення складових умов використовуються оператори булевої алгебри: "і", "або"

Оператори

- && та
- || або

```
if (t<100 && t>=0) {
console.log("Це вода");
}
```

```
if ((year%4==0 && year%100!=0) || year%400==0){
console.log("Рік - високосний");
}
```

Розділ 4. Цикли.

4.1. Загальний опис циклів

У програмуванні ми можемо часто зустріти такі алгоритми, які вимагають повторити ту саму ділянку коду двічі, тричі і більше разів. При цьому часто змінюються тільки параметри, які стоять на вході даного фрагмента коду, що повторюється.

Наприклад:

1. Вивести 7 разів слово "Привіт!".
2. Звести число до 5-го ступеня.
3. Вивести у стовпчик числа від 1 до 8.

З одного боку, якщо ми скопіюємо та вставимо код у програмі, то отримаємо бажаний результат. Однак при цьому довжина програми буде занадто великою, і, якщо ми захочемо внести хоча б 1 зміну в частині, що повторюється, або змінити кількість повторень, нам доведеться вносити зміни вже в код програми.

При цьому простою дією "копіювання-вставка" практично неможливо вирішити задачу, в якій кількість повторень задається ззовні (вводиться користувачами або виходить від сервера). Щоб вирішувати завдання такого класу, використовують спеціальну конструкцію – «цикли». Розглянемо найпростіший приклад:

Таблиця 1. Програми-відповідники із використанням структури циклів та без неї

Програма без використання циклів	**Програма із використанням циклів**
`console.log("Привіт");` `console.log("Привіт");` `console.log("Привіт");` `console.log("Привіт");` `console.log("Привіт");`	`for(let i = 0;i<6;i++)` `  console.log("Привіт");`

`let a = prompt("Введіть основу ступеня ");`
`let b = 1;`
`b *= a;`
`b *= a;`
`b *= a;`
`b *= a;`
`b *= a;`
`alert("Введене число 5-го ступеня = " + b);`
	`let a = prompt("Введіть основу ступеня ");`
	`let b = 1;`
	`for (let i=0;i<5;i++)`
	`   b *= a;`
	`alert("Введене число 5-го ступеня = " + b);`

`console.log(1);`
`console.log(2);`
`console.log(3);`
`console.log(4);`
`console.log(5);`
`console.log(6);`
`console.log(7);`
	`for (let i = 1; i < 8; i++)`
	`    console.log(i);`

4.2. Види циклів.

Існує 3 основних види циклів: з параметром, з передумовою та з постумовою. Наведемо приклад реалізації різних циклів на простій задачі – виведення на екран чисел від 1 до 10.

4.2.1. Цикл із параметром

Цикл з параметром – цикл, у якому змінюється одна змінна (параметр чи «лічильник циклу»). На початку циклу змінної параметру задається початкове значення, яке змінюється на кожному кроці циклу (на 1). Цикл виконується до того моменту, поки значення параметра менше або дорівнює кількості закінчення циклу. Найчастіше цикли з параметром застосовують коли необхідно переглянути числову послідовність, наприклад, від 1 до 10.

```
for (let i = 1; i <= 10; i++) {
     console.log(i);
 }
```

4.2.2. Цикл із передумовою

Цикл із передумовою – цикл, у якому умова виконання перебуває до описи операторів («тіла циклу»). Цикл з передумовою найчастіше застосовується у випадках, коли умова закінчення циклу залежить від операторів у тілі циклу.

```
let i = 0;
let sum = 0;
while (sum <= 100) {
   console.log(sum);
   i = Number(prompt("Введите число"));
   sum += i;
}
```

4.2.3. Порівняння з циклом із параметром

Цикл з параметром - це, по суті, окремий випадок циклу з передумовою, в якому збільшення параметра закладено в структурі циклу і завдяки зручному запису для його реалізації необхідно менше рядків коду.

```
 let i = 1;
 while (i <= 10) {
     console.log(i);
     i++;
 }
```

```
for (let i = 1; i <= 10; i++) {
     console.log(i);
}
```

4.2.4. Цикл із постумовою

Цикл з постумовою – цикл, у якому умова виконання перебуває після опису операторів. Такий цикл виконається щонайменше один раз у будь-якому випадку (навіть якщо умова виконання буде хибною).

```
let userPass = '';
let currentPass = 'password';
do {
    userPass = prompt("Введите пароль");
} while (userPass != currentPass);
console.log("Вы авторизованы");
```

4.2.5. Порівняння з циклом із параметром

Цикл з постумовою також може реалізувати той же функціонал, що і цикл з параметром, але він займатиме більше рядків коду.

```
let i = 1;
do {
    console.log(i);
    i++;
} while (i <= 10);
```

Кожен цикл має свої особливості та відповідний клас завдань, для яких він призначений.

4.3. Загальна структура циклу

- Дія на першому етапі циклу (для циклу з параметром це означає присвоєння змінній-лічильнику початкового значення; для циклу з передумовою – значення, які ми задаємо до початку циклу змінним, які використовуються в тілі циклу).
- Умова виконання циклу.
- Дія кожного кроку циклу (оператори в тілі циклу)

4.4. Базові алгоритми для структури «цикли»

Для циклів такими базовими алгоритмами є:

1. Обчислення суми чисел.
2. Обчислення добутку чисел.
3. Обчислення середнього арифметичного (середнього геометричного) значення ряду чисел.
4. Дії зі значеннями за заданою ознакою.
5. Визначення наявності елементів, які відповідають заданій умові.
6. Пошук найбільшого (найменшого) із введених чисел.
7. Робота циклу до виконання умови

4.4.1. Обчислення суми чисел

Даний алгоритм використовується у тих ситуаціях, у яких необхідно порахувати

суму ряду чисел, що вводяться. При цьому введення може здійснюватися користувачем (з клавіатури), файлу, випадковим чином (за допомогою функції Math.random();) або ж значення можуть бути результатом математичної формули, пов'язаної з лічильником циклу.

Особливістю даного алгоритму є змінна-накопичувач, в якій ми будемо акумулювати суму, додаючи до неї черговий доданок на кожному кроці.

циклу.

Змінна-накопичувач перед початком циклу, який обчислює суму, **повинна мати значення 0**, тому що при складанні першого доданку з нулем ми в результаті отримаємо перший доданок. Зазвичай змінну-накопичувач для суми називають **sum**, або **s**.

```
let s = 0;
for (let i = 1; i <= 10; i++)
    s += i;
console.log('Сумма чисел от 1 до 10 равна '+ s);
```

4.4.2. Обчислення добутку

Алгоритм обчислення добутку чисел найчастіше застосовується у завданнях з математики для обчислення різних функцій (ступінь, факторіал та ін.), з теорії ймовірностей, а також з економіки (обчислення підсумкового значення ціни за наявності низки змін, обчислення підсумкової суми вкладу, ін.) .

Алгоритм обчислення твору схожий на алгоритм обчислення суми, з тією лише відмінністю, що змінна-накопичувач до початку циклу, який обчислює добуток чисел, має значення 1, оскільки перший співмножник, помножений на 1,дасть у змінній-накопичувачі себе ж (якби ми залишили першим значенням 0, то результатом завжди був би 0, тому що будь-яке число, помножене на 0, дає у творі 0). Змінну-накопичувач для твору зазвичай називають mult (скорочення від англійського слова «multiplication» – множення):

```
let mult = 1;
for (let i = 1; i <= 10; i++) {
    mult *= i; //mult = mult * i;
}
console.log("Добуток чисел від 1 до 10 = " + mult);
```

4.4.3. Обчислення середнього значення

Дуже великий клас завдань – це обчислення різних середніх значень (у математиці середні значення часто називають просто «середні»): середній бал атестата, середня успішність зі школи, середній рівень доходу протягом року, середній темпи зростання країни.

Ми розберемо циклічний алгоритм, використовуючи який можна швидко обчислювати середні для будь-якої кількості даних, що вводяться.

Алгоритм обчислення середнього значення ґрунтується на алгоритмі обчислення суми (для обчислення середнього арифметичного значення) та на алгоритмі твору (для обчислення середнього геометричного значення). До даних алгоритмів додається обчислення кількості введених значень. Для позначення змінної, де обчислюється середнє, часто використовується ім'я avg (скорочення від англійського слова «average» – середній).

Для позначення змінної, яка зберігає кількість чисел використовують ім'я n. Ця змінна збільшується на 1 при додаванні нового елемента до послідовності. У першому прикладі значення змінної n в

Наприкінці циклу буде співпадати зі значенням змінної-лічильником циклу. Це окремий випадок, що зустрічається досить рідко, тому ми відразу обчислюємо значення змінної n окремо від лічильника циклу.

Середнє арифметичне	**Середнє геометричне**

`let sum = 0;`
`let n = 0;`
`for (let i = 1; i <= 10; i++) {`
`  sum += i; //sum = sum + i;`
`  n += 1;`
`}`
`let avg = sum / n;`
`console.log('Середнє арифметичне чисел від 1 до 10 = ' + avg);`
	`let mult = 1;`
	`let n = 0;`
	`for (let i = 1; i <= 10; i++) {`
	`  mult *= i;`
	`  n += 1;`
	`}`
	`let avg = Math.pow(mult, 1 / n);`
	`console.log('Середнє геометричне чисел від 1 до 10 = ' + avg);`

4.4.4. Алгоритм «Дії зі значеннями за заданою ознакою»

Цей тип алгоритмів є складовим. У його реалізації беруть участь як цикл, і розгалуження (алгоритм вибору). Суть роботи даного алгоритму зводиться до того, що у циклі генерується необхідна послідовність чисел (наприклад, числа від 1 до 10), а розгалуження вибирає ті числа, які відповідають заданій ознаці.

Таке додавання (додавання розгалуження) може застосовуватися до всіх вивчених базових алгоритмів. При додаванні розгалуження можна обчислювати не всі числа від 1 до 10, а, наприклад, лише парні. Той же принцип і з середніми: можна обчислювати середнє арифметичне не всіх чисел, що вводяться, а тільки великих заданого значення. Одним із найпростіших прикладів на даний алгоритм є обчислення суми парних чисел:

Сума парних чисел від 1 до 10

```
let sum = 0;
for (let i = 0; i <= 10; i++) {
	if (i % 2 == 0) {
	sum += i;
	}
}
console.log('Сумма парних чисел от 1 до 10 дорівнює ' + sum);
```

Сума парних із 10 випадкових

```
let sum = 0;
for (let i = 0; i <= 10; i++) {
	let a = Math.ceil(Math.random() * 10);
	console.log(a);
	if (a % 2 == 0) {
		sum += a;
	}
}
```

```
console.log('Сума парних з 10 випадкових чисел дорівнює ' + sum);
```

У повсякденному житті ми часто стикаємося з такою ситуацією, в якій нам цікаво не загальне значення послідовності даних (сума або твір), не кількість елементів заданої ознаки, а чи є хоча б один елемент заданої ознаки. Такі ситуації відбуваються, коли оцінюється та чи інша група.

Наприклад: При здачі нормативів бойовим підрозділом залік йде за останнім

(тобто якщо хоча б один боєць норматив не здав, «незалік» ставиться всьому підрозділу). Подібний підхід спостерігаємо на митниці: якщо документи видано на групу з 3-х машин і хоча б одна з них не відповідає заданим параметрам, то через кордон не буде пропущено всю групу.

У програмуванні такі завдання вирішуються за допомогою прапорної змінної. Прапорна змінна змінює значення 1 раз під час зустрічі елемента, який задовольняє (не задовольняє) заданому ознакою.

Виходить, що наша прапорна змінна набуває значення або «Брехня» (ознака немає), або «Істина» (ознака є). Так само поводяться змінні типу boolean. Логічно на початку циклу прапорної змінної надати значення False, а при «спрацьовуванні ознаки» змінити значення на True. У цьому випадку ми зможемо вставити ім'я змінної прямо за умови розгалуження.

"Завдання про нормативи в армії".

У програму вводяться нормативи бійців із бігу за хвилини. Норматив для проходження дистанції (марш-кидок на 5 км) – 30 хвилин. Якщо хоча б один із солдатів не укладається у встановлений час, норматив вважається не зданим. Перевірити, чи виконав підрозділ поставлене перед ним завдання

```
let n = prompt('Введіть кількість воїнів у підрозділі');
let flag = false;
for (let i = 1; i <= n; i++) {
	let a = prompt('Введіть норматив №' + i);
	if (a > 30) {
		flag = true;
	}
}
if (flag) {
	alert('Залік не зданий')
} else {
	alert('Залік зданий');
}
```

У той самий час можна сказати, що ми отримали значення хоча б одного елемента, який відповідає ознакою, то вивчення подальших елементів не впливає на кінцевий результат. Тобто. цикл можна «зупинити»: цей хід результат не змінить, зате вивільнить багато часу.

Для таких випадків у циклах використовують оператор break; Цей оператор зупиняє виконання циклу.

```
let n = prompt('Введіть кількість воїнів у підрозділі');
let flag = false;
for (let i = 1; i <= n; i++) {
	let a = prompt('Введіть норматив №' + i);
	if (a > 30) {
		flag = true;
		break;
	}
}
if (flag) {
	alert('Залік не зданий')
} else {
	alert('Залік зданий');
```

```
}
```

4.4.5. Алгоритм «Пошук найбільшого із введених чисел»

Цей алгоритм складається з наступних кроків:

1. Змінної, в якій обчислюється максимум, надається значення першого елемент.
2. Далі у циклі значення змінної для максимуму порівнюється зі значеннями всіх елементів послідовності.
3. Якщо елемент послідовності буде меншим, ніж змінна для зберігання максимуму, то значення даного елемента надається змінною для зберігання максимуму.
4. Після виконання циклу змінна для зберігання максимуму виводиться на екран

Алгоритм для пошуку мінімального значення послідовності аналогічний (відмінності: при порівнянні змінна зберігання мінімуму отримує значення, якщо поточний елемент менше її значення).

Максимум

```
let max = 0;
/*
 змінної max присвоюється мінімальне з можливих випадкових
чисел за даної форми завдання
*/
for (let i = 1; i <= 5; i++) {
    let a = Math.ceil(Math.random() * 10);;
    console.log(a);
     if (a > max) {
        max = a;
    }
}
console.log('Максимальне значення = ' + max);
```

Минимум

```
let min = 11;
/*
змінної min присвоюється максимальне з можливих випадкових
чисел за даної форми завдання
 */
for (let i = 1; i <= 5; i++) {
    let a = Math.ceil(Math.random() * 10);;
    console.log(a);
    if (a < min) {
min = a;
    }
}
console.log('Мінімальне значення = ' + min);
```

4.5. Генерування послідовностей:

Цикли також використовуються для генерації послідовностей, наприклад: вивести всі високосні роки за XX століття, вивести всі числа, менші за 100, кратні 5 і т.д. Генерація послідовності часто є складовою різних завдань.

Для створення послідовності нам необхідно:

- початкове значення послідовності;
- кінцеве значення або умова закінчення послідовності;
- алгоритм (умова), яким ми відбираємо «потрібні» елементи.

Ця нескладна дія дозволяє вирішувати дуже цікаві завдання.

Де це застосовується?

Наприклад, нам необхідно сформувати список усіх високосних років за XVIII, XIX, XX, XXI сторіччя.

Це завдання цікава, з одного боку, тим, що у ній генерується послідовність, з іншого – тим, що послідовність генерується з допомогою серії розгалужень, т.к. Умова, чи рік високосний, задається за допомогою трьох правил:

1. Високосним роком вважається той рік, номер якого ділиться на 4 без залишку.
2. Якщо номер року без залишку ділиться також на 100, то рік не є високосним (наприклад, 1900 не є високосним).
3. Якщо номер року без залишку ділиться і на 100, і на 400, то рік є високосним (наприклад, 2000 є високосним).

Перший крок – ми визначаємо початкове значення послідовності (у нашому це перший рік XVIII століття або 1800 рік).

Другий крок – ми визначаємо кінцеве значення послідовності (у нашому випадку це останній рік XXI століття або 2099).

Третій крок – ми задаємо умову відбору (у разі це буде складова умова).

У нашому випадку умова така:

((n % 4 ==0)&& !(n %100 = 0))||(n % 400 = 0)

```
for (let i = 1800; i <= 2099; i++) {
     if (((i % 4 == 0) && !(i % 100 == 0)) || (i % 400 == 0)) {
        console.log(i);
     }
}
```

Розділ 5. Масиви

5.1. Загальний опис масивів

Масиви – це особливий вид змінних, який для одного ідентифікатора (імені) зберігає не 1 значення, як ми звикли, працюючи зі звичайними змінними, а кілька (стільки, скільки ми вкажемо при завданні масиву). При цьому всі елементи масиву відносяться до одного типу даних, а ми можемо робити з кожним елементом ті ж дії, що і змінної відповідного типу.

Якщо провести аналогію, що змінна – це скринька, в яку ми можемо покласти тільки одне значення, тоді масив – це скринька з секціями, і в кожну його секцію можна покласти лише одне значення. Всі секції скриньки-масиву пронумеровані, починаючи з 1 (або з тієї цифри, яку ми визначили як номер першого елемента).

Як відбувається робота з масивами? Так само, як і зі змінними: ми можемо надавати їм значення, можемо ці значення виводити на екран, значення елементів масиву можуть брати участь у визначенні значень інших змінних і т.д.

Розглянемо з прикладу роботу з масивами. Припустимо, нам необхідно встановити випадковим чином 5 чисел, збільшити їх на 1 і вивести результат на екран. Порівняємо два варіанти програми: перший – з використанням змінних, другий – з використанням масиву

Таблиця 2. Програми-відповідники при використанні масивів та без нього

Змінні	**Масиви**
`let a = Math.ceil(Math.random() * 10);`	`let aa = Array();`

```
let b = Math.floor(Math.random() * 10);
let c = Math.floor(Math.random() * 10);
let d = Math.floor(Math.random() * 10);
let e = Math.floor(Math.random() * 10);

a += 1;
b += 1;
c += 1;
d += 1;
e += 1;

console.log(a + " " + b + " " + c + " " + d
+ " " + e);
```

```
aa[0] = Math.floor(Math.random() * 10);
aa[1] = Math.floor(Math.random() * 10);
aa[2] = Math.floor(Math.random() * 10);
aa[3] = Math.floor(Math.random() * 10);
aa[4] = Math.floor(Math.random() * 10);

aa[0] += 1;
aa[1] += 1;
aa[2] += 1;
aa[3] += 1;
aa[4] += 1;

console.log(aa[0] + " " + aa[1] + " " +
aa[2] + " " + aa[3] + " " + aa[4]);
```

Найкраще зрозуміти зручність масивів виходить з прикладу їх використання разом із циклами. Коли навіть при 5 елементах масиву кількість рядків коду зменшується майже 2 рази.

```
let aa = Array();
let res = "";
for (let i = 0; i < 5; i++) {
     aa[i] = Math.floor(Math.random() * 10);
     aa[i] += 1;
     res += aa[i] + " ";
}
console.log(res);
```

5.2. Оголошення та завдання значень масиву

Для того щоб з масивом працювати його необхідно ініціалізувати та привласнити йому необхідні значення.

Введення значень випадковим чином

```
let aa = Array();
for (let i = 0; i < 5; i++) {
    aa[i] = Math.floor(Math.random() * 10);
}
```

Введення значень у вигляді числової послідовності

```
let aa = Array();
  for (let i = 0; i < 5; i++) {
    aa[i] = i;
}
```

Введення значень користувачем

```
let aa = Array();
for (let i = 0; i < 5; i++) {
   aa[i] = prompt('Введіть ' + i + '-й елемент масива');
}
```

Наступний алгоритм, можливо, ще простіший, ніж алгоритм завдання масиву – це виведення значень масиву в консоль

Виведення значень у консоль

```
/*...*/
let aa = Array();
for (let i = 0; i < 5; i++) {
```

```
    aa[i] = Math.floor(Math.random() * 10);
}

/*...*/
       let res = "";
for (let i = 0; i < 5; i++) {
    res += aa[i] + " ";
}
console.log(res);
```

Програма, яка демонструє введення значень елемента масиву різними способами та виведення їх у консоль.

```
let aa = Array();
let res = "";

/* Введення значень випадковим чином */
for (let i = 0; i < 5; i++) {
    aa[i] = Math.floor(Math.random() * 10);
}

/* Виведення значень у консоль */
for (let i = 0; i < 5; i++) {
    res += aa[i] + " ";
}
console.log(res);
/* Введення значень у вигляді числової послідовності */
for (let i = 0; i < 5; i++) {
    aa[i] = i;
}

/* Виведення значень у консоль */
let res = "";
for (let i = 0; i < 5; i++) {
    res += aa[i] + " ";
}
console.log(res);
/* Виведення значень у консоль */
for (let i = 0; i < 5; i++) {
    aa[i] = prompt('Введіть ' + i + '-й елемент масива');
}

/* Виведення значень у консоль */
let res = "";
for (let i = 0; i < 5; i++) {
    res += aa[i] + " ";
}
console.log(res);
```

Наочне представлення масиву.

Для наочності можна записати масив у вигляді таблиці з одного рядка

2	3	4	5	3	2	1	2	3	5

Над кожною коміркою – напишемо її порядковий номер, який є індексом елемента масиву.

Індекс елемента	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Значення	2	3	4	5	3	2	1	2	3	5

елемента										

Таким поданням масиву ми користуватимуться надалі, коли вивчатимемо різні дії, що проводяться з масивів.

5.3. Завдання на роботу із масивом

1. Масив має вигляд

2	**3**	**4**	**5**	**4**	**3**	**2**	**1**	**5**	**6**

Задайте його у програмі.

2. Елементами масиву із 6 елементів є ступеня (починаючи з першої) числа 2, задайте значення для кожного елемента масиву.

2	**2*2**	**2*2*2**	**2*2*2*2**	**2*2*2*2*2**	**2*2*2*2*2*2**

3. Виконайте завдання №2 за допомогою циклу.

4. Елементи масиву із п'яти елементів, задані за наступною закономірністю:

rnd	**[0]*2**	**[1]+1**	**[0]*[1]**	**[1]*[3]**

где rnd – число, заданное случайным образом, [1] – первый элемент массива, [2] – второй элемент массива, т.д.

5.4. Пошук елемента масиву, який відповідає заданим параметрам.

Після того, як ми навчилися задавати елементи масиву і виводити їх на екран, нам необхідно навчитися працювати з елементами масиву. Адже масив у програмі – це не що інше, як відображення у пам'яті комп'ютера того чи іншого набору даних (наприклад: список середніх балів атестату навчальної групи, результати спортсменів на змаганнях тощо).

А дані послідовності нам потрібні для того, щоб ми отримували від них ті чи інші дані – це може бути середнє арифметичне за всіма елементами (середній бал групи); відповідь на запитання – чи є в масиві елемент із заданим значенням (наприклад – стартовий номер того чи іншого спортсмена) та на якому місці він знаходиться; або чи виконує хоча б один елемент певним умовам (чи є в класі хлопці вище 180 сантиметрів).

Також вкрай важливими і найчастіше використовуваними під час роботи з масивами є алгоритми пошуку мінімального і максимального елементів, і навіть алгоритми сортування (упорядкування) елементів масиву.

Ми розглядатимемо кожен алгоритм, його основні принципи та приклад роботи. Для кожного алгоритму будуть дані практичні завдання, які Вам необхідно виконати, щоб натренувати здатність роботи з масивами.

Наприкінці даного уроку ми розглянемо набір завдань на роботу з масивами, при цьому будуть дані вхідні та правильні вихідні дані для цих завдань – щоб Ви могли перевірити себе самостійно.

5.5. Розрахунок середнього арифметичного елементів масиву.

Для реалізації даного алгоритму нам доведеться переглянути всі елементи масиву і на кожному кроці перегляду - збільшувати змінну-накопичувач суми на значення елемента, вважатимемо, що кількість елементів в масиві - ми знаємо.

Ми вже з Вами розглядали в уроці №2 обчислення середнього арифметичного, але для зберігання елементів ми використовували змінні. Робота з елементами масиву набагато зручніша, адже ми можемо переглянути будь-яку кількість елементів масиву, написавши в програмі всього два рядки: цикл та рядок для обчислень.

І якщо в уроці №2 ми розглядали обчислення середнього арифметичного значення набору змінних, скоріше як навчальний приклад (на практиці таку реалізацію зустріти практично неможливо), то обчислення середнього значення для масиву – це популярне завдання у повсякденній роботі програміста.

Будь-який алгоритм починається із введення даних і закінчується виведенням даних, т.к. якщо ми не введемо дані - то нам не буде з чим працювати, а якщо ми не виведемо дані - то ми не дізнаємося результат. При цьому крім самого результату (власне середнього арифметичного) буде правильно виводити всі елементи масиву на екран (бажано виводити елементи масиву в рядок, тому що в рядку міститься більше елементів у вікні виводу, ніж при виведенні в стовпець).

Тобто. схема нашого алгоритму буде виглядати так:

Розглянемо практичний приклад розрахунку середнього арифметичного значення масиву.

```
let aa = Array();
// Завдання масиву
for (let i = 0; i <= 5; i++) {
    aa[i] = Math.floor(Math.random() * 10);
}
// Виведення елементів масиву в рядок
let str = '';
for (let i = 0; i <= 5; i++) {
    str += aa[i] + ' ';
}
console.log(str);

let sum = 0;
let n = 0;
for (let i = 0; i < aa.length; i++) {
    sum += aa[i];
    n += 1;
}
let avg = sum / n;
console.log('Середнє арифметичне елементів масиву = ' + avg);
```

5.6. Пошук мінімального елемента у масиві.

Алгоритм пошуку мінімального елемента в масиві аналогічний алгоритму пошуку максимального, за винятком того, що ми в змінну-накопичувач «складаємо» значення елемента масиву, коли воно менше поточного.

```
let min = aa[0]
for (i = 1; i < aa.length; i++) {
     if (aa[i] < min) min = aa[i];
}
console.log(min);
```

5.7. Пошук місця знаходження мінімального/максимального елемента у масиві.

Але для вирішення деяких завдань необхідно не просто знайти значення максимального або мінімального значення в масиві, а знати на якому місці цей елемент розташований. Для вирішення такого завдання нам необхідно вдосконалити попередню задачу: коли змінна-накопичувач буде більше/менше поточного значення елемента масиву ми запам'ятовуємо не тільки значення такого елемента, але і його місце в масиві, для цього, природно, використовуємо додаткову змінну.

```
let aa = Array();
 // Завдання масиву
for (let i = 0; i <= 5; i++) {
    aa[i] = Math.floor(Math.random() * 10);
}
 // Виведення елементів масиву в рядок
let str = '';
for (let i = 0; i <= 5; i++) {
     str += aa[i] + ' ';
}
console.log(str);
 let min = aa[0]
 let imin = 0;
for (i = 1; i < aa.length; i++) {
   if (aa[i] < min) {
       min = aa[i];
       imin = i;
   }
}
console.log("Мінімальний елемент масиву " + min + " Знаходиться на позиції " + imin);
```

5.8. Пошук місця елемента, який відповідає певній умові.

Пошук максимально і мінімального елемента масиву, а також визначення місця такого елемента, хоча і є дуже важливим і поширеним завданням при роботі з масивами, але все ж таки є окремим випадком більш загального завдання.

Адже максимальний елемент – це просто елемент, який відповідає певному критерію (найбільше решти елементів масиву). Але в реальних завданнях умови пошуку елемента можуть бути різними, при цьому суть вирішення таких завдань – не зміниться. Наприклад: «При прийомі працювати всі кандидати проходять тестування.

У масиві aa задані результати проходження тестування всіх кандидатів у хронологічній послідовності (на першому місці стоїть результат кандидата, який проходив тестування найпершим). Знайти номер кандидата по порядку, якщо відомо, що фірма прийняла першого кандидата, який отримав у процесі тестування результат 170 (з 200) та вище».

```
let aa = Array();
for (let i = 1; i <= 20; i++) {
aa[i] = Math.floor(Math.random() * 50) + 150;
console.log(aa[i]);
     }
for (let i = 1; i <= aa.length; i++) {
   if (aa[i] >= 170) {
        console.log('Кандидат під номером ', i, ' перший пройшов співбесіду');
        break;
   }
}
```

5.9. Сортування масивів

Цей клас завдань передбачає зміну елементів масиву, тобто. перестановка їх відповідно до вибраного параметра. Найчастіше це сортування за зростанням або за зменшенням значень. Інші варіанти зустрічаються набагато рідше, але вони також мають місце (наприклад: можна відсортувати масив за кратністю чисел або за значенням суми цифр).

Для того, щоб сортування «відбулося», нам необхідно здійснити певну послідовність дій: виконати алгоритм сортування. Таких алгоритмів є досить багато. Ми розглянемо найпростіші з них, а також познайомимося з критерієм оцінки ефективності алгоритмів сортування, а також торкнемося високоефективних алгоритмів сортування.

Місця застосування сортування.

А навіщо взагалі щось із масивом робити, що не можна просто взяти потрібні нам елементи? Це питання, яке перевіряє на міцність важливість вивчення алгоритмів сортування. Він має право на існування, тим більше що тема сортувань масивів досить об'ємна і досить складна і без достатньої мотивації до вивчення та подальшого застосування алгоритмів сортування – навряд чи буде освоєна повною мірою.

Сортування застосовується практично у будь-якій сфері роботи комп'ютерів. Починаючи з найпростішого – ви можете натиснути правою клавішею миші на вільному полі вашого Робочого столу та вибрати сортування за типом, розміром, датою зміни, типом елемента. Всі ці дії будуть виконані за допомогою певних алгоритмів сортування.

В електронних таблицях Excel є ціла функція "Сортування". Ви можете відсортувати комірки у стовпцях та рядках так, як Ви оберете.

У пошукових системах застосовуються алгоритми сортування. І хоча в пошукових системах питання підрахунку значення параметра є набагато складнішим питанням, ніж сортування результатів пошуку по них, проте алгоритми сортування в них також займають значуще місце.

Як ми переконалися – сортування потрібне! Тепер познайомимося з тим, як працюють алгоритми сортування.

5.10. Алгоритм сортування вибором.

Перший алгоритм сортування, який ми розглянемо, ґрунтується на алгоритмі пошуку максимального елемента, який ми розглянули вище. Тому нам буде легко освоїти його.

Для початку визначимо ті поняття, якими ми будемо оперувати щодо даного алгоритму:

Не відсортована частина масиву (НВЧМ) – та частина масиву, в якій поки що не проводилося сортування.

Відсортована частина масиву (ВЧМ) – частина масиву, в якій елементи відсортовані.

Алгоритм обміну значень – алгоритм, у якому беруть участь три змінні (наприклад: a, b, k). В результаті роботи алгоритму змінна a матиме початкове значення змінної b і навпаки змінна b матиме початкове значення змінної

a.

```
k=a; //Привласнюємо проміжною змінною k значення змінної a
 // (Зберігаємо a)
a = b; //надаємо в змінній a значення змінної b (затираємо a)
b = k; //Привласнюємо змінною a значення проміжної змінної k
 //(Відновлюємо a)
```

Розглянемо цей алгоритм на такому прикладі: ми розклали собі 6 фотографій і хочемо їх відсортувати в хронологічному порядку. Час, коли зроблено дані фотографії (у місяцях, від сьогоднішньої дати) розподілено так.

50	20	5	10	100	1

Ми вибираємо найновішу фотографію і хочемо поставити її на початок. Але для того, щоб продовжувати бачити всі фотографії – ми змінюємо її місцями з тією, що лежала на місці найновішої фотографії (фотографії, зробленої місяць тому).

50	20	5	10	100	1

1	20	5	10	100	50

Далі – розглядаємо НВЧМ та шукаємо знову в ньому мінімальне значення. Тобто. Наступна фотографія – це фотографія, зроблена 5 місяців тому. Ми її міняємо місцями із тією, яка лежить на другому місці.

1	20	5	10	100	50

1	5	20	10	100	50

1 20 5 10 100 50

1 5 20 10 100 50

На наступному кроці – ми знову шукаємо ту фотографію, яка зроблена у термін, який найближчий до сьогоднішнього дня. На цьому кроці – ми бачимо, що елемент із мінімальним значенням цільового показника – вже стоїть на потрібному місці.

1	5	10	20	100	50

1	5	10	20	100	50

Для фотографій, що залишилися, повторюємо описаний алгоритм. Фотографію з терміном створення 50 місяців ми поставимо 5 місце, а на шосте, відповідно, пересунеться фотографія з терміном створення 100 місяців

1	5	10	20	100	50

1	5	10	20	50	100

Загальний опис алгоритму:

Знаходимо максимальний елемент.

Змінюємо його місцями із першим.

Тепер у нас з'явилася ВЧМ – перший елемент є максимальним.

Повторюємо пп. 1,2,3 для НВЧМ, що залишилася, поки ВЧМ не стане рівною за розміром всьому масиву.

```
let aa = Array();
 //Завдання масиву
 for (let i = 0; i <= 5; i++) {
 aa[i] = Math.floor(Math.random() * 10); // Завдання елементів масиву випадковим чином
 }

 let str = '';
 for (let i = 0; i <= 5; i++) {
 str += aa[i] + '';
 }
 console.log(str); // Виведення елементів масиву в рядок.
 for (let i = 0; i < aa.length; i++) {
 max = aa [i]; // Задаємо змінної max перший елемент
 // Невідсортована частина масиву (НВЧМ)
 jmax = i; // задаємо змінною jmax номер першого елемента НВЧМ.
  for (let j = i + 1; j < aa.length; j++) { // запускаємо цикл від 2-го елемента НВЧМ до
кінця
 //Массива.
 if (aa[j] > max) { //Якщо зустрічаємо елемент більше max, то

 max = aa[j]; //Змінною max привласнюємо значення цього елемента
 jmax = j; //Змінною jmax присвоюємо значення цього номера
 //елемента
 }
 }

 k = aa[i]; //Змінюємо місцями
 aa[i] = aa[jmax]; //перший елемент НВЧМ
 aa[jmax] = k; //з максимальним елементом НВЧМ
 }

 str = '';
 for (let i = 0; i <= 5; i++) {
 str += aa[i] + '';
 }
 console.log(str); // Виведення елементів масиву в рядок.
```

5.11. Асоціативні масиви

Асоціативний масив - це масив у якого як індекси використовуються рядки.

```
let ar = Array();
ar["txt"] = "main";
console.log(ar["txt"]);
```

Розділ 6. Робота з елементами Document Object Model (DOM)

Мова програмування JavaScript дозволяє обробляти дії, які робить користувач на сторінці.

Для того, щоб запустити певну дію, необхідно визначити ту подію, після якої відбуватиметься ця дія. Найпростіша подія – це завантаження сторінки. Саме такою подією ми користувалися у всіх попередніх завданнях.

Тобто програма

```
<script>
  console.log("Hello!");
</script>
```

запускається за умови завантаження сторінки.

Решта варіантів - це події, які відбуваються з елементами на сторінці. І ці події слід описувати.

Наступний приклад – при натисканні кнопки з'являється сервісне поле із повідомленням "Введіть число". Після введення номера з'являється повідомлення з тим це парне чи непарне число.

```
<button id="btn">Click me!</button>
 <script>
 btn.addEventListener("click", fnc); /* на елемент з id=btn підключається "слухач", який за події "click" запускається функція fnc */

 function fnc() { //функція fnc
 console.log("fnc");
 let a = prompt("Введіть число");
 if (a % 2 == 0)
 alert("Число парне");
 else
 alert("Число непарне");
 }
</script>
```

Важливим питанням, яке необхідно враховувати під час роботи з будь-якою програмою, у т.ч. та JavaScript це введення та виведення. У попередній програмі ми читали дані із сервісного вікна та у сервісне вікно виводили.

Насправді використання сервісного вікна відбувається рідко. Найчастіше використовується введення даних у блоці елементів введення (input).

```
   <div>a: <input type="text" id=a></div>
    <div>b: <input type="text" id=b></div>
    <div>result: <input type="text" id=res></div>
    <button id="btn">Click me!</button>
    <script>
btn.addEventListener("click", fnc);

function fnc() {
    let va = Number(a.value);
    let vb = Number(b.value);
    let vres = va + vb;
    res.value = vres;
}
```

Згенерувати можна будь-який елемент. При цьому елементу можна встановити практично будь-яке значення його властивостей.

Для того, щоб згенерувати елемент використовується команда document.createElement('element_name'); де замість element_name вставляється назва тега. Для того, щоб згенерувати елемент div необхідно вказати document.createElement('div'); Для того, щоб згенерувати абзац - document.createElement('p');

Але сама собою генерація елемента нічого не дає. Для того, щоб можна було працювати зі згенерованим елементом, ми повинні його зробити значенням певної змінної.

```
let elem = document.createElement('p');
console.log(elem);
```

Перший рядок даного елемента створює елемент p і надає його змінної elem. Другий рядок - виводить цей елемент на консоль.

Після генерації самого тега часто необхідно внести в нього той чи інший текст або HTML-код. Для цього використовуються методи innerText та innerHTML відповідно.

```
let elem = document.createElement('p');
elem.innerHTML='<b>Text</b>';
console.log(elem);
```

Даний змінної elem надасть значення згенерованого елемента p, привласнить цьому елементу HTML-текст Text і виведе результат у консоль.

Але при створенні елемента за допомогою методу document.createElement, присвоєння значення створеного елемента змінної та роботи з нею всі дії відбуваються в оперативній пам'яті. Для того, щоб вивести створений елемент на екран, використовуються методи prepend і append. Перший метод (prepend) вставляє створений елемент доти, до якого застосовується метод, а другий (append) - після.

HTML	**JS**

`<div id ="some_div">--Some Text--</div>`	`let elem = document.createElement('p');`
	`elem.innerHTML='<b>Text</b>';`
	`some_div.append(elem);`

Даний приклад додасть елемент elem всередину елемента з ідентифікатором some_div після існуючого елемента тексту або елементів

HTML	**JS**

`<div id ="some_div">--Some Text--</div>`	`let elem = document.createElement('p');`
	`elem.innerHTML='<b>Text</b>';`
	`some_div.prepend(elem);`

Даний приклад додасть елемент elem всередину елемента з ідентифікатором some_div до існуючого елемента тексту або елементів

Додавати створені елементи в DOM можна також за допомогою методів appendChild і insertBefore

Також за допомогою js можна задавати стилі оформлення елементів:
Напряму прописати значення CSS за допомогою властивості style.cssText

HTML	`<div id ="some_div">--Some Text--</div>`

JS	`let elem = document.createElement('p');`
	`elem.innerHTML='<b>Text</b>';`
	`elem.style.cssText  = "color:red;background:blue;";`
	`some_div.append(elem);`

Додати певний клас

HTML	`<div id ="some_div">--Some Text--</div>`

CSS	`.blue{`
	`  color:blue;`
	`}`
	`.decoration{`
	`  text-decoration:underline;`
	`}`

JS	`let elem = document.createElement('p');`
	`elem.innerHTML='<b>Text</b>';`
	`elem.classList  = "blue decoration";`
	`some_div.append(elem);`

Розділ 7. Функції

Часто трапляється так, що одна написана ділянка колись зручно використовувати кілька разів. Такий підхід дозволяє зменшити розмір програм, а також спростити їхню зміну.

Найпростіша аналогія з функціями у програмуванні – це функції в математиці. Наприклад, функція вилучення квадратного кореня з числа 2 у мові програмування JavaScript буде записана як Math.sqrt(2);

```
let a = Math.sqrt(2);
console.log(a);
```

Math - це математична бібліотека JavaScript. Вона включає такі функції як:

- Math.abs(x) - модуль
- Math.ceil(x) - округлення до більшого
- Math.cos(x) - косинус
- Math.floor(x) - округлення до меншого
- Math.pow(x,y) - зведення x до ступеня y
- Math.random() - псевдовипадкове число від 0 до 1
- Math.round(x) - округлення за правилами математики
- Math.sqrt(x) - квадратний корінь
- Math.sin(x) - синус

Але крім певних функцій існує можливість задавати функції користувача. Зміст функцій користувача задає автор коду.

Наприклад, такий код виведе 6 разів сервісне вікно зі словом Hello

```
function hll(){
  alert('Hello');
  alert('Hello');
}

function hll();
function hll();
function hll();
```

Крім простого повторення дій функції дозволяють отримувати параметри та використовувати їх у обчисленнях усередині функції

```
function plus (a,b){
    let c = a+b;
    console.log(c);
}
plus (2,3);
```

Крім того, що функція може приймати певні параметри і робити з ними обчислення, також функція може повертати результат, що обчислюється. Саме такий формат функції максимально відповідає математичним функціям.

```
function minus (a,b){
    let c = a-b;
    return c;
}
let d = minus(4,3);
console.log(d);
```

Розділ 8. Об’єктно-орієнтоване програмування

8.1. Класи

Важливим елементом будь-якої мови програмування є така структура як клас. Якщо спрощено, то класи - це така структура, яка поєднує в собі асоціативні масиви та функції.

Найпростішим прикладом використання класу – це завдання певної структурованої змінної, яка зберігає дані у тому числі різного типу.

```
class student{
    fio = "Name";
    age = 21;
    avgBall = 89;
}
let st = new student();
console.log(st.fio);
st.fio = "T";
console.log(st.fio);
console.log(st.age);
console.log(st.avgBall);
```

Крім прямого завдання значень за замовчуванням можна використовувати інструмент ООП як конструктор. Це метод, який запускається під час ініціалізації об'єкта (у прикладі st) певного класу (у прикладі student)

```
class student {
    constructor(name) {
    this.name = name;
}
}
let st = new student("Petro");
console.log(st.name);
```

8.2. Методи

Важливою можливістю класів є можливість задавати внутрішні функції об'єкта класу (методи).

```
class student {
```

```
    constructor(name) {
    this.name = name;
    this.age = 21;
}
 getName(){
    return this.name;
}
setName(n){
    this.name = n;
}
upAge(){
    this.age+=1;
}
getAge(){
    return this.age;
}

}
let st = new student("Petro");
console.log(st.name);
st.setName("Max");
console.log(st.getName());
st.upAge();
st.upAge();
console.log(st.getAge());
```

8.3. JSON

Для обміну даними між різними програмами використовуються різні формати. На сьогодні основним форматом у якому обмінюються дані програми написані у тому числі різними мовами є JSON.

Зручність даного формату полягає в тому, що стандартними інструментами мови рядка цього формату створюється об'єкт і навпаки.

Переведення об'єкта в рядок:

```
class student {
    fio = "Name";
    age = 21;
    avgBall = 89;
}
let st = new student();
st.fio = "Max";
let json = JSON.stringify(st);
console.log(json);
```

Переклад рядка у форматі JSON на об'єкт:

```
let json = '{"fio":"Max","age":21,"avgBall":89}';
let st = JSON.parse(json);
console.log(st.fio);
```

Отримання даних.

Дані можна отримати безпосередньо від програм, написаних розробником певної інформаційної системи, але дуже часто дані беруться з відкритих джерел.

Форматом надання даних є публічні API (інтерфейс надання даних).

Наприклад, відкритий API Приватбанку: https://api.privatbank.ua/

За запитом: https://api.privatbank.ua/p24api/pubinfo?json&exchange&coursid=5

Ми отримаємо відповідь у форматі JSON, де ми отримуємо поточне значення курсів валют. Наприклад:

[{"ccy":"USD","base_ccy":"UAH","buy":"26.65000","sale":"27.05000"},{"ccy":"EUR","base_ccy":"UAH
","buy":"28.85000","sale":"29.45000"},{"ccy":"BTC","base_ccy":"USD","buy":"8469.4379","sale":"9360.9577"}]