

# Веб- розробка

## I РОЗДІЛ. СТВОРЕННЯ ВЛАСНИХ ВЕБ-СТОРІНОК. (стр. 4 – 20)

- 1.1. З чого складається веб-сторінка. (стр. 4)
- 1.2. Що таке HTML. (стр. 4)
- 1.3. Невізуальні компоненти. (стр. 4 – 6)
- 1.4. Написання тексту. (стр. 6 – 8)
- 1.5. Атрибути. (стр. 9 – 10)
- 1.6. Додавання кнопок, прапорців, перемикачів, тощо. (стр. 10 – 12)
- 1.7. Додавання зображень. (стр. 13)
- 1.8. Створення таблиць. (стр. 13)
- 1.9. Створення нумерованих та маркірованих списків. (стр. 14)
- 1.10. Створення випадajuчих списків. (стр. 15)
- 1.11. Додавання відео та аудіо контенту. (стр. 15 – 16)
- 1.12. Використання блоків: <div>; <footer>; <header>; тощо. (стр. 16 – 19)
- 1.13. Викладення сайту у мережу. (стр. 19 – 20)

## II РОЗДІЛ. СТИЛІЗАЦІЯ СТОРІНОК. (стр. 21 – 49)

- 2.1. Що таке CSS. (стр. 21)
- 2.2. Використання CSS на HTML. (стр. 21 – 24)
- 2.3. Стилiзація класів та id. (стр. 24)
- 2.4. Загальні опції. (стр. 25 – 30)
- 2.5. Display-опція. (стр. 31 – 33)
- 2.6. Фільтри. (стр. 33 – 36)
- 2.7. Flexbox. (стр. 36 – 44)
- 2.8. Анімація. (стр. 44 – 47)
- 2.9. Адаптація. (стр. 47 – 49)

## III РОЗДІЛ. СКРИПТИ. (стр. 50 – 71)

- 3.1. Що таке JavaScript. (стр. 50)
- 3.2. Додавання та виконання скриптів на веб-сторінці. (стр. 50 – 51)
- 3.3. Події. (стр. 51 – 55)
- 3.4. Перші скрипти. (стр. 55 – 58)
- 3.5. Оператори, умови, розгалуження та цикли. (стр. 59 – 62)
- 3.6. Функції. (стр. 63)
- 3.7. Рандом. (стр. 64 – 65)
- 3.8. Скрипти для веб-програмування та фреймворки. (стр. 66 – 71)

## IV РОЗДІЛ. ВИКОРИСТАННЯ БАЗИ ДАНИХ. (стр. 72 – 90)

- 4.1. Що таке база даних. (стр. 72)
- 4.2. Реляційні та нереляційні бази даних. (стр. 72)
- 4.3. Використання JSON за допомогою JavaScript. (стр. 73)
- 4.4. Що таке SQL. (стр. 74)
- 4.5. Отримання даних, сортування, умови та оператори. (стр. 75 – 90)

Цей посібник пов'язан з веб-програмуванням. У цій книжці ви навчитесь: створення веб-сторінок; стилізація сторінок; програмування мовою JavaScript; використовувати базу даних за допомогою мовою програмування SQL. Ці мови дуже популярні у веб-розробці та першим з чого починають вивчати веб-розробники – використання HTML, який є на усіх веб-сторінках.

# І РОЗДІЛ. СТВОРЕННЯ ВЛАСНИХ ВЕБ-СТОРІНОК.

## Урок 1.1. З чого складається веб-сторінка.

Загалом сторінки складаються з елементів, стилізації та функціональності. За об'єкти відповідає мова гіпертекстової розмітки HTML, за стилізацію відповідає мова каскаду стилів CSS, а за функціональністю відповідають скрипти мов програмування.

## Урок 1.2. Що таке HTML.

HTML (Hyper Text Marker Language) – мова гіпертекстової розмітки, яка відповідає за наявність елементів на сайті. Ось написання коду на HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- <head> - тег, який відповідає за наявність
           елементів функцій. Де lang="en" - вказування
           на мову сторінки, щоб браузер розумів, якою
           мовою там написано.
           <!DOCTYPE html> - вказування використання
           версії HTML (у даному випадку, це: HTML 5.0)
    -->
  </head>
  <body>
    <!-- <body> - тег, який відповідає за наявність
           візуальних елементів -->
  </body>
</html>
```

Де писалось `<!-- якийсь текст -->` - це коментар, який не впливає на роботу сторінки.

Також треба пам'ятати, що: `<html>`; `<body>`; `<head>` - це все теги. Вони можуть бути з закриваючим тегом та без нього. Наприклад, теги: `<body>`; `<head>`; `<html>` - відкриваючі теги, а теги, які мають на початку своєї назви символ « / », такі як: `</head>`; `</body>`; `</html>` - закриваючі теги. Про закриваючі та відкриваючі теги не треба забувати.

## Урок 1.3. Невізуальні компоненти.

Загалом невізуальні компоненти це ті, які ми не бачимо на веб-сторінці, але виконують функцію, такі як: імпорт скриптів; зміна загальної назви сторінки; збереження інформації про сторінку; тощо. Для використання

невізуальних компонентів, треба їх використовувати у тезі <head>. Ось код з використанням цих компонентів:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Встановлення загальної
    назви сторінки -->
    <title>Назва сторінки</title>

    <!-- Імпорт стилізації -->
    <link rel="stylesheet" href="файлСтилюСторінки.css">

    <!-- Задання іконки сайту -->
    <link rel="icon" type="image/png" href="./assets/favicon.png">

    <!-- Інформація про сторінку, яку можна знайти за
    ключовими словами в атрибуті "content". Де атрибут
    "name" у якому значення "keywords" - це вказання на
    виконання функції тегу <meta>. У даному випадку -
    було вказано про функцію пошуку за ключовими словами
    за пошуковою системою-->
    <meta name="keywords" content="вікіпедія, Метатег, стаття">

    <!-- Інформація про ім'я автора сторінки -->
    <meta name="author" content="Вадим Савенко">

    <!-- Аналог "author" де можна ще вказати мову -->
    <meta name="copyright" lang="en" content="Вадим Савенко">

    <!-- Інформація про сторінку (Його опис) -->
    <meta name="description" content="Сторінка HTML">

    <!-- Інформація про програму, якою використовувалось для
    створення даної веб-сторінки -->
    <meta name="generator" content="Macromedia Dreamweaver 4.0">

    <!-- Інформація про мову сторінки -->
    <meta http-equiv="content-language" content="en">

    <!-- Інформація про -->
    <meta http-equiv="Content-Style-Type" content="text/stylus-lang">
    <!-- можна написати таким чином -->
    <meta http-equiv="Content-Style-Type" content="text/less">

    <!-- Інформація про кодування сторінки -->
    <meta charset="UTF-8">
    <!--Продовження коду у наступній сторінці -->
```

```

<!-- Інформація про те, що дану сторінку треба відкривати
у фреймі -->
<meta http-equiv="Window-target" content="_top">

<!-- ICBM: дозволяє прив'язати всю сторінку до географічних
координат -->
<meta name="ICBM" content="12.345, 67.890">

<!-- Позначте групу geo. виконує аналогічну задачу, дозволяючи
явно вказати назву населеного пункту і регіон -->
<meta name="geo.position" content="50.167958;-97.133185">
<meta name="geo.placename" content="Manitoba, Canada">
<meta name="geo.region" content="ca-mb">
</head>
<body>
  <!-- <body> - тег, який відповідає за наявність
візуальних елементів -->
</body>
</html>

```

Це усі невізуальні компоненти, які потрібно знати кожному, але були ще показані усі обов'язкові компоненти. Ось код з обов'язковими компонентами:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Назва сторінки</title>
    <link rel="stylesheet" href="файлСтилюСторінки.css">
    <link rel="icon" type="image/png" href="favicon.png">
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- <body> - тег, який відповідає за наявність
візуальних елементів -->
  </body>
</html>

```

**Примітка!** Якщо не використовувати <title>, то загальна назва сторінки буде мати назву файлу коду цієї сторінки.

## Урок 1.4. Написання тексту.

Для сторінки, текст – візуальний компонент, який має декілька символів (Це: літери; крапка; кома; крапка з комою; двокрапка; тощо). Код з тегами тексту є у наступній сторінці.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- <head> - тег, який відповідає за наявність
         елементів функцій (невізуальних компонентів).-->
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- Звичайний текст -->
    <p>
      р р р р р.
      р р р р р.
    </p>

    <!-- Звичайний текст -->
    <pre>
      pre pre pre.
      pre pre pre.
    </pre>

    <!-- Заголовок 1-ого рівня -->
    <h1>Ця сторінка про текст.</h1>

    <!-- Заголовок 2-ого рівня -->
    <h2>Ця сторінка про текст.</h2>

    <!-- Заголовок 3-ого рівня -->
    <h3>Ця сторінка про текст.</h3>

    <!-- Заголовок 4-ого рівня -->
    <h4>Ця сторінка про текст.</h4>

    <!-- Заголовок 5-ого рівня -->
    <h5>Ця сторінка про текст.</h5>

    <!-- Заголовок 6-ого рівня -->
    <h6>Ця сторінка про текст.</h6>

    <!-- Гіперпосилання. Де дані у "href",
         там посилання на сторінку -->
    <a href="https://website/">Ця сторінка про текст.</a>

    <!-- Жирний текст -->
    <b>Ця сторінка про текст.</b>
    <!-- Продовження коду у наступній сторінці -->
```

```

<br> <!-- Переводє елементи та текст на іншу строку -->

<!-- Наклонений текст -->
<i>Ця сторінка про текст.</i>

<!-- Підкреслений текст -->
<u>Ця сторінка про текст.</u>

<hr> <!-- Створює смугу -->

<!-- Закреслений текст -->
<s>Ця сторінка про текст.</s>

<!-- Підстрочний текст -->
<sub>Ця сторінка про текст.</sub>

<!-- Надстрочний текст -->
<sup>Ця сторінка про текст.</sup>
</body>
</html>

```

У разі перевірки коду у браузері, буде різниця між тегами:

р р р р р. р р р р р.

pre pre pre.  
pre pre pre.

# Ця сторінка про текст.

## Ця сторінка про текст.

### Ця сторінка про текст.

#### Ця сторінка про текст.

##### Ця сторінка про текст.

###### Ця сторінка про текст.

Ця сторінка про текст. Ця сторінка про текст.

*Ця сторінка про текст.* Ця сторінка про текст.

---

~~Ця сторінка про текст.~~ Ця сторінка про текст. Ця сторінка про текст.



## Урок 1.5. Атрибути.

Атрибут – ємна, необхідна для забезпечення цілісності об'єкта, суб'єкта (предмета) властивість, його частина, додаток. Атрибути - це поля даних, які відносяться до файлу, але не є його частиною. Вони не враховуються при підрахунку розміру файлу і можуть бути скопійовані або видалені без зміни самого файлу. Система використовує атрибути для зберігання, наприклад, розміру, типу файлу, дати його останньої зміни тощо. Так само й у в інших операційних системах. Відмінність полягає в тому, що Ви можете додати будь-який атрибут до будь-якого файлу, потім відобразити і зробити його таким, що можна редагувати у вікні Tracker. Вам потрібно лиш визначити якого типу атрибут Ви бажаєте додати до файлу (наприклад: стрічка, ціле число і час), дати визначення і опис. Питання для чого потрібний атрибут → він вказує шлях, або маршрут браузеру, атрибут вказує тегу куди розмістити текст, який повинен бути список тощо. Які бувають атрибути → конкретний який потрібний для конкретного тегу, і загальний або спільний, який використовується для абзацу, для заголовку, для таблиці. Ну що ж в принципі все, є атрибути які можна додавати до будь-якого тегу, а є конкретні які додають до конкретного тегу, добавлю що атрибут “align” вважають застарілим, але так можна сказати що сама мова гіпертекстової розмітки HTML застаріла, і не потрібна коли присутні системи управління контентом.

Атрибути завжди записуються всередині тегу, після них іде знак рівності і деталі атрибуту, укладені в подвійні лапки. Кривка з комою після атрибуту служить для поділу команд різних стилів. Ось така записується тег з атрибутами:

```
<тег атрибутПерший="" атрибутДругий=""></тег>
```

Примітка: тег може мати декілька атрибутів. Атрибути не є обов'язковими для використання з тегами, але є деякі атрибути, які треба використовувати для функціональності або стилізації тегу. Ось усі атрибути, які використовуємо у тегах:

**Атрибути, які є усіх візуальних компонентах:** class; id; style; width; . Для прикладу візьмемо тег <p>. Ось написання цього тегу з використання цих атрибутів:

```
<p class="class_p" id="id_p" style="color: red;">Hello</p>
```

*class* – служить для задавання класу для тегу.

*id* – служить для задавання ID для тегу.

*style* – служить для задавання CSS стилю для тегу.

*width* – служить для задання широти компонента у процентах від розміру сторінки у вікні браузера або у пікселях. Наприклад: `width="10px"`; `width="10%"`.

*align* – служить для виставлення компонент горизонтально з центрального (`align="center"`), правого (`align="right"`) або лівого боку (`align="left"`).

**Атрибути *src* та *href*:** атрибути, які служать для посилання на локальний файл (У тому числі і сторінки) або мережевий файл:

```

```

Різниця між атрибутами *src* та *href*:

- *src* використовується у тегах: `<img>`; `<iframe>`; `<video>`; `<source>`.
- *href* використовується у тегах: `<a>`; `<meta>`; `<link>`.

Для посилання на адресу, треба вказувати таким чином:

- **На веб-сторінку:**

```
href="https://на.веб.сторінку/"
```

- **На електронну пошту:**

```
href="mailto:mailuser2000@mail.com"
```

- **На номер телефона:**

```
href="tel:+380931112233"
```

**Атрибути *alt*:** атрибут, який вставляє текст замість зображення, якщо зображення не вдалося завантажити. Цей тег використовується лише з тегом `<img>`. Ось його написання з цим тегом:

```

```

**Атрибути *target*:** атрибут, який використовується лише у тезі `<a>` та відкриває посилання у новій вкладці, якщо `target="_blank"`. Наприклад:

```
<a href="google.com" target="_blank">Link</a>
```

**Атрибути *download*:** атрибут, який використовується лише у тезі `<a>` та дає команду браузеру, що посилання на файл можна лише завантажувати. Наприклад:

```
<a href="google.com/file.txt" download="file.txt">Link</a>
```

**Атрибути *type* та інші:** про ці атрибути буде розмова в інших потім.

## Урок 1.6. Додавання кнопок, прапорців, перемикачів, тощо.

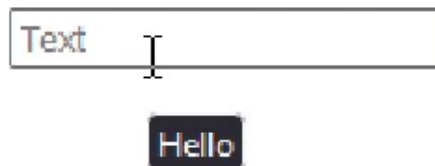
Тег `<button>` відповідає за створення кнопки, яка відповідає за виконання функції при натискання на неї. Ось написання коду з цим тегом:

```
<button>Кнопка</button>
```

Також є тег `<input>`, який відповідає за створення поля вводу. Ось написання коду з цим тегом:

```
<input title="Hello" placeholder="Text" type="text">
```

У цьому випадку на сторінці буде:

A screenshot of a web form. It features a single-line text input field with a light gray border. Inside the field, the word 'Text' is written in a light gray font, serving as a placeholder. A vertical cursor is positioned at the end of the text. Directly below the input field is a small, dark gray rectangular button with the word 'Hello' written in white text.

Але з цим тегом можна робити не тільки кнопки, а ще й інші компоненти за допомогою атрибуту `type`. Ось приклади використання з атрибутом `type`:

type	Компоненти
“text”	Текстове поле
“email”	Поле поштової адреси
“tel”	Поле номера телефону
“button”	Кнопка
“checkbox”	Прапорець
“number”	Числове поле
“date”	Поле дати
“time”	Поле часу
“datetime-local”	Поле дати та часу
“password”	Поле пароллю
“color”	Кнопка для обрання кольору
“file”	Кнопка для обрання файлу
“radio”	Перемикач (Радіо-кнопка)
“range”	Повзунок

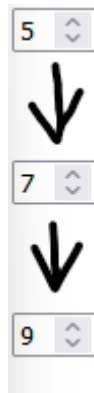
Також тег `<input>` має необов’язкові атрибути:

- **min** – мінімальна допустима кількість символів або значення у компоненті
- **max** – максимальна допустима кількість символів або значення у компоненті
- **step** – на скільки шагів буде переміщуватись повзунок чи числове поле.

Використання їх у кодї сторінки:

```
<input min="5" max="9" step="2" type="number">
```

Результат коду у браузері:

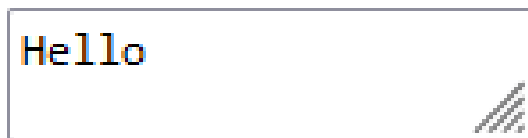


Ці атрибути можна використовувати з типами: “time”; “range”; “number”.

Також є тег `<textarea>`, який відповідає за створення поля вводу, у якому можна змінювати його розмір. Ось його написання у кодї:

```
<textarea title="Hi">Hello</textarea>
```

При відкриття сторінки у браузері:



Також у цього тега є атрибут «rows», який відповідає за висоту цього тега за строками. Якщо не вказувати цей атрибут, то його висота буде дорівнювати двум строкам. Нехай буде випадок, коли треба зробити висоту цього компонента буде на 5 строк. Тоді, ось приклад при використанні цього атрибута:

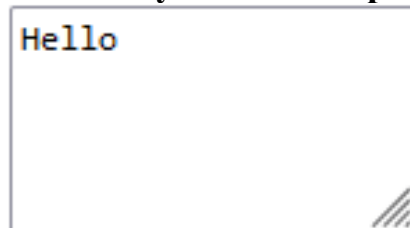
```
<textarea rows="5">Hello</textarea>
```

При відкриття сторінки у браузері:

**Якщо не вказувати цей атрибут**



**Якщо вказувати цей атрибут**



## Урок 1.7. Додавання зображень.

Для додавання зображень на сторінку HTML, треба використовувати тег `<img>`, який має атрибути `alt` та `src`. Ось код з цим тегом:

```


```

- `alt` служить для вставлення тексту замість зображення у випадок, коли завантажити зображення було невдалою спробою.
- `src` служить для задання зображення для тегу `<img>`.

## Урок 1.8. Створення таблиць.

Ось код для створення таблиці на сторінці:

```
<table>
  <tbody>
    <tr>
      <th>1 стовбець</th>
      <th>2 стовбець</th>
      <th>3 стовбець</th>
    </tr>
    <tr>
      <th>4</th>
      <th>5</th>
      <th>6</th>
    </tr>
    <tr>
      <th>7</th>
      <th>8</th>
      <th>9</th>
    </tr>
  </tbody>
</table>
```

Ось що буде на сторінці:

**1 стовбець 2 стовбець 3 стовбець**

**4**

**5**

**6**

**7**

**8**

**9**

## Урок 1.9. Створення нумерованих та маркірованих списків.

Для додавання зображень на сторінку HTML, треба використовувати тег `<ol>` для нумерованих списків та `<ul>` для створення маркірованих списків:

```
<ol>
  <li><p>1 елемент</p></li>
  <li><h3>2 елемент</h3></li>
  <li>3 елемент</li>
</ol>
<ul>
  <li><p>1 елемент</p></li>
  <li><h3>2 елемент</h3></li>
  <li>3 елемент</li>
</ul>
```

Ось що буде на сторінці:

1. 1 елемент
  2. **2 елемент**
  3. 3 елемент
- 1 елемент
  - **2 елемент**
  - 3 елемент

Також можна задавати для нумерованих списків атрибут *start*, який відповідає з якого номеру починається елемент. Наприклад:

```
<ol start="7">
  <li>1 елемент</li>
  <li>2 елемент</li>
  <li>3 елемент</li>
</ol>
```

Ось що буде на сторінці:

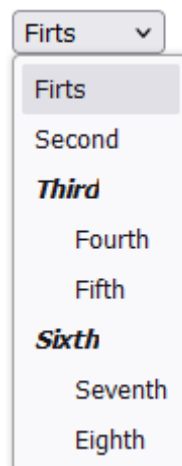
7. 1 елемент
8. 2 елемент
9. 3 елемент

## Урок 1.10. Створення випадаючих списків.

Для додавання зображень на сторінку HTML, треба використовувати тег `<select>`:

```
<select name="name" id="id">
  <option>Firts</option>
  <option>Second</option>
  <optgroup label="Third">
    <option>Fourth</option>
    <option>Fifth</option>
  </optgroup>
  <optgroup label="Sixth">
    <option>Seventh</option>
    <option>Eighth</option>
  </optgroup>
</select>
```

Ось що буде на сторінці:



## Урок 1.11. Додавання відео та аудіо контенту.

Для додавання аудіо-файлів використовуємо тег `<audio controls>`, а для додавання відео використовуємо тег `<video controls>`. Ось приклад у цьому коді:

```
<!-- Додавання аудіо-файлів -->

<!-- Додавання формату .mp3 -->
<audio controls>
  <source src="audio.mp3" type="audio/mp3">
</audio>

<!-- Продовження коду у наступній сторінці -->
```

```

<!-- Додавання формату .wav -->
<audio controls>
  <source src="audio.wav" type="audio/wav">
</audio>

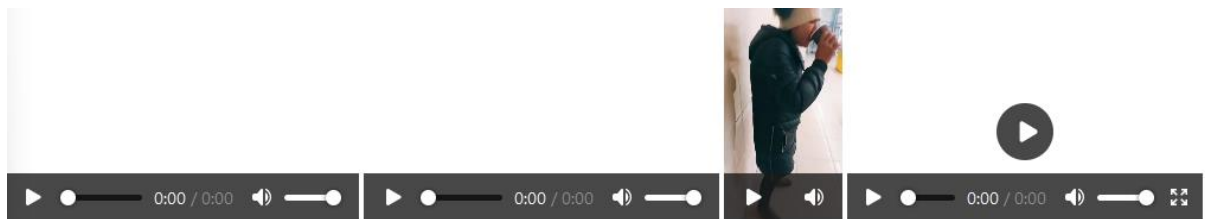
<!-- Додавання відео -->

<!-- Додавання формату .mp4 -->
<video controls>
  <source src="video.mp4" type="video/mp4">
</video>

<!-- Додавання формату .avi -->
<video controls>
  <source src="video.avi" type="video/avi">
</video>

```

Ось що буде на сторінці:



Також можна додавати відео з YouTube-сервісу. Треба перейти на потрібне відео та клікнути на кнопку «Поділитися» та обрати «Вставити», де буде тег `<iframe>` з атрибутами та цей код треба вставити до нашого HTML-коду.

## Урок 1.12. Використання блоків: `<div>`; `<footer>`; `<header>`; тощо.

HTML-елементи `<div>`, `<footer>`, та `<header>` широко використовуються для структурування і оформлення веб-сторінок. `<div>` (від "division") - це універсальний контейнер, який використовується для групування інших HTML-елементів. Він не має жодного спеціального значення, але часто застосовується для організації структури сторінки та застосування CSS-стилів. Ось приклад у цьому коді:

```

<div style="width: 200px;">
  <h1>Header Header Header</h1>
  Main Content
</div>
<!-- Продовження коду у наступній сторінці -->

```



```
<div style="width: 200px;">
  <h1>Header Header Header</h1>
  Main Content
</div>
```

Ось що буде на сторінці:

**Header**  
**Header**  
**Header**

Main Content

**Header**  
**Header**  
**Header**

Main Content

Але `<div>` - це єдиний контейнер у HTML, бо є ще: `<nav>`; `<footer>`; `<header>`; `<section>`; `<article>`; `<aside>`; `<main>`; `<figure>`; `<figcaption>`; `<dialog>`.

- `<nav>`: Використовується для визначення навігаційних посилань. Наприклад, меню сайту.

```
<nav>
  <ul>
    <li><a href="#home">Головна</a></li>
    <li><a href="#about">Про нас</a></li>
    <li><a href="#contact">Контакти</a></li>
  </ul>
</nav>
```

- `<footer>`: Представляє нижній колонтитул секції або сторінки. Часто містить авторські права, посилання на політику конфіденційності та іншу інформацію.

```
<footer>
  <p>©2024 Компанія. Всі права захищені.</p>
</footer>
```

- `<header>`: Використовується для представлення введення або групи навігаційних посилань. Зазвичай включає логотип, заголовок або навігаційні посилання.

```
<header>
  <h1>Заголовок Сайту</h1>
  <nav>
    <ul>
      <li><a href="#home">Головна</a></li>
      <li><a href="#about">Про нас</a></li>
      <li><a href="#contact">Контакти</a></li>
    </ul>
  </nav>
</header>
```

- `<section>`: Використовується для визначення розділів у документі, таких як глави, вкладки тощо.

```
<section>
  <h2>Про Нас</h2>
  <p>Тут можна додати інформацію про вашу компанію.</p>
</section>
```

- `<article>`: Представляє автономний фрагмент контенту, який може бути розповсюджений незалежно. Наприклад, блог пост або новинна стаття.

```
<article>
  <h2>Заголовок Статті</h2>
  <p>Текст статті...</p>
</article>
```

- `<aside>`: Використовується для вмісту, який побічно пов'язаний з основним контентом. Наприклад, бокова панель з посиланнями або рекламою.

```
<aside>
  <h2>Бокова Панель</h2>
  <p>Додатковий контент або реклама тут.</p>
</aside>
```

- `<main>`: Використовується для основного вмісту документа. Повинно бути лише одне `<main>` на сторінку.

```
<main>
  <h1>Головний Заголовок</h1>
  <p>Основний контент вашої сторінки.</p>
</main>
```

- `<figure>`: Використовується для позначення незалежного вмісту, такого як ілюстрація, діаграма або фото, разом з описом.

```
<figure>
  
  <figcaption>Підпис до зображення</figcaption>
</figure>
```

- `<figcaption>`: Використовується для додавання підпису до `<figure>`.

```
<figure>
  
  <figcaption>Підпис до зображення</figcaption>
</figure>
```

- `<dialog>`: Використовується для створення діалогового вікна або модального вікна.

```
<dialog>
  <p>Це діалогове вікно.</p>
  <button>Закрити</button>
</dialog>
```

### Урок 1.13. Викладення сайту у мережу.

Сервіс для завантаження сайту, створений зі HTML-сторінок, найліпший та безкоштовний, це – GitHub. Для початку треба зареєструватись. Після реєстрації у цьому сервісі, треба створити публічний репозиторій з файлом README. Після створення репозиторію, треба натиснути на «Add file» та на «Upload files», де буде перехід на додавання файлів. Після переходу на сторінку для завантаження, треба завантажити усі файли, які взаємопов'язані з HTML-сторінками. Далі треба перейти на «Setting» та перейти на «», де буде випадаючий список, у якому обрано «None». Треба замість «None» обрати «main» та натиснути на кнопку «Save» та почекати деякий час (Приблизно, це 5 хвилин). Після очікування деякого часу треба перезавантажити сайт, у якому повинно з'явитись нове посилання на створений сайт.

Але це лише завантаження сайту у мережу, а якщо треба, щоб користувач зміг знайти сайт через пошукову систему (), то треба ще під'єднати сторінку до пошукової системи. Ось довідки, як додати посилання на сайт до пошукової системи Google (Посилання:

<https://support.google.com/webmasters/>) та для Bing (Посилання: <https://www.bing.com/webmasters/help>).

Це не усі HTML-теги, які були показані у цій надбавці, але це є базові теги, які треба знати. Усі теги є на сайті [css.in.ua](https://css.in.ua), де є усі теги, скрипти JavaScript та CSS компоненти для стилізації тег. Там також показано усі атрибути для кожного тегу. Ось посилання на сайт:

<https://css.in.ua/html/tags>. Також там показано, які теги можна використовувати лише у версії HTML 5.0, які можна використовувати у всіх версіях HTML (Крім 5.0, бо вони у цій версії не підтримуються), та які підтримуються на усіх версіях HTML.

## II РОЗДІЛ. СТИЛІЗАЦІЯ СТОРІНОК.

### Урок 2.1. Що таке CSS.

Якщо HTML служить для розміщення елементів на сторінці, то CSS (Cascade Style Sheet) відповідає за їхню стилізацію. Мова CSS теж не є мовою програмування.

### Урок 2.2. Використання CSS на HTML.

Для використання стилізації у HTML-сторінках можна робити 3 шляхами:

1. **Вбудований стиль (Використання атрибуту *style*):** у атрибуті *style* можна вписувати опції для тегу. Наприклад:

```
<div style="стиль тегу"></div>
```

2. **Вбудована таблиця стилів (Використання тегу *<style>*):** у тезі *<style>* можна вписувати селекторів та їхні опції для тег, id та класів у сторінці. Він може бути у тезі *<body>* або *<head>*. Наприклад:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: blue;
      }
    </style>
  </head>
  <body>
    <p>Hello</p>
  </body>
</html>
```

3.

4. **Зовнішня таблиця стилів (Імпорт файлу “.css”):** використовуємо тег *<link>*. Наприклад:

```
<link rel="stylesheet" href="файлСтилюСторінки.css">
```

Задання стилю складається з:

```
селектор {
  опція: значення;
}
```

Також можна робити значення через змінні:

```
селектор {
  --зміння: значення
  опція: var(--зміння);
  опціяДва: var(--зміння);

  /* При цьому у двох опціях буде однакове значення */
}
```

До речі! Де у коді: «/\* При цьому у двох опціях буде однакове значення \*/», ця строка є - **коментар**.

Нехай ми робимо стилізацію для цього коду:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <p>Hello</p>
  </body>
</html>
```

Тепер на треба зробити, щоб наш тег <p> мав синій шрифт на 14 пікселів. Тоді треба нам казати селектор «p», бо треба це зробити для тегу <p>. Ось таким чином ми будемо робити у коді «style.css»:

```
p {
  /* Задання кольору */
  color: blue;

  /* Задання розміру шрифту у пікселях */
  font-size: 14px;
}
```

Тепер на сторінці побачимо:

Hello

Також можна зробити через змінні, спосіб який теж зробить такий-же результат, як перший:

```
p {
  --color: blue;
  --size: blue;
  color: var(--color);
  font-size: var(--size);
}
```

Можна також не вказувати назву кольору у значенні, а використовувати функцію RGB.

```
p {
  color: rgb(xxx, yyy, zzz);

  /* Або таким чином */
  color: #XXYYZZ;
  /*
  XX, xxx - Червоний відтінок
  YY, yyy - Зелений відтінок
  ZZ, zzz - Синій відтінок
  */
}
```

Також є розуміння, як – **псевдо-класи**. Ось їхнє використання у таблиці стилю:

```
селектор:псевдо-клас {
  опція: значення;
}
```

Псевдо-клас – стан селектору.

Селектор – це загалом теги, id (#id) та класи (.class).

Ось таблиця усіх базових псевдо-класів:

Псевдо-клас	Умова
hover	Коли користувач натиснув курсором на селектор лівою кнопкою мишки
focus	Коли користувач натиснув курсором на селектор правою кнопкою мишки
active	Коли користувач натиснув та не відпускає ліву кнопку мишки на селектор
visited	Коли користувач натиснув на цей селектор та перейшов на сторінку (Загалом використовують це у тезі <a>)

Ось приклад використання псевдо-класів:

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

### Урок 2.3. Стилiзацiя класiв та id.

Маємо код, який має теги: <h1>; <h2>; <p>, які мають клас «cls» та id «i».  
Ось і він:

```
<!DOCTYPE html>  
<html>  
    <head>  
        <link rel="stylesheet" href="style.css">  
    </head>  
    <body>  
        <h1 class="cls" id="i">Hello</h1>  
        <h2 class="cls" id="i">Hello</h2>  
        <p class="cls" id="i">Hello</p>  
    </body>  
</html>
```

Тепер треба зробити так, щоб усі текстові теги були червоними. Можна зробити так:

```
h1, h2, p {  
    color: red;  
}
```

Можна й таким чином, але якщо вони мають спільний клас або id, то можна зробити так:

```
/* Для класів */  
.cls {  
    color: red;  
}  
  
/* Для id */  
#i {  
    color: red;  
}
```



## Урок 2.4. Загальні опції.

Ось код с загальні опції у CSS:

```
селектор {
  /* Основи стилізації елементів */

  /* Задає колір тексту */
  color: red;

  /* Визначає сімейство шрифту */
  font-family: 'Segoe UI';

  /* Розмір шрифту (У пікселях) */
  font-size: red;

  /* Товщина шрифту (Число) */
  font-weight: 100; /* 100 - Тонкий, найменша товщина */
  font-weight: 400; /* 400 - Стандартний */
  font-weight: 600; /* 600 - Жирний */

  /* Стиль шрифту (italic, normal або bold) */
  font-style: italic; /* Наклонний стиль шрифту */
  font-style: normal; /* Звичайний стиль шрифту */
  font-style: bold; /* Жирний стиль шрифту */

  /* Оздоблення тексту (підкреслення, перекреслення) */
  text-align: red;

  /* Оздоблення тексту (підкреслення, перекреслення) */
  text-decoration: underline; /* підкреслення */
  text-decoration: overline; /* перекреслення */

  /* Зміна регістру букв */
  text-transform: uppercase; /* Усі букви стануть великими */
  text-transform: capitalize; /* Перша літера кожного слова буде
                               великою */
  text-transform: lowercase; /* Усі букви стануть маленькими */
  text-transform: none; /* Залишає текст без змін, як є за
                          замовчуванням */
  text-transform: inherit; /* Успадковує властивість від
                            батьківського елемента */
  text-transform: initial; /* Значення за замовчуванням, яке
                            визначене браузером */
  text-transform: unset; /* Скидає властивість до значення, яке
                          залежить від контексту */

  /* Продовження коду у наступній сторінці */
}
```

```
/* Висота рядка тексту (У пікселях) */
line-height: 1.5px;

/* Інтервал між символами (У пікселях) */
letter-spacing: 0.5px;

/* Інтервал між словами (У пікселях) */
word-spacing: 2px;

/* Прозорість */
opacity: 0.7; /* 70% прозорість */

/* Рівень розмиття */
filter: blur(5px); /* Розмиття на 5 пікселів */

/* Основи макетування */

/* Визначає тип блока для HTML-елемента
(block, inline, flex, grid) */
display: block; /* Блочний елемент */
display: inline; /* Рядковий елемент */
display: flex; /* Гнучкий контейнер */
display: grid; /* Сітковий контейнер */

/* Встановлює ширину елемента (У пікселях) */
width: 10px;

/* Встановлює висоту елемента (У пікселях) */
height: 10px;

/* Зовнішні відступи навколо елемента (У пікселях) */
margin: 10px;

/* Внутрішні відступи навколо вмісту елемента (У пікселях) */
padding: 10px;

/* Встановлює кордон елемента (ширина, стиль, колір) */
/* Кольор */
border: red;
/* Ширина (У пікселях) */
border: 10px;
/* Стиль */
border: none; /* Відсутність кордону */
border: currentColor; /* Колір кордону дорівнює
                      кольору тексту */
border: dashed; /* Кордон складається з коротких
                штрихів */

/* Продовження коду у наступній сторінці */
```

```
border: dotted; /* Кордон складається з точок */
border: double; /* Подвійна лінія кордону */
border: groove; /* Виглядає як вирізаний у фоні */
border: hidden; /* Кордон прихований (еквівалентно
                  'none') */
border: inherit; /* Спадкує стиль кордону */
border: inset; /* Виглядає як втоплений у фоні */
border: initial; /* Ініціалізація до значення за
                  замовчуванням */
border: medium; /* Середній розмір кордону (еквівалентно
                  Зрх за замовчуванням) */
border: outset; /* Виглядає як випуклий над фоном */
border: ridge; /* Виглядає як гребінь на фоні */
border: solid; /* Пряма неперервна лінія кордону */
border: thick; /* Товщий кордон */
border: thin; /* Тонший кордон */
border: unset; /* Скидання до значення за замовчуванням */

/* Відступи від країв батьківського елемента для
   позиціонованих елементів (У пікселях) */
top: 10px; /* Відступ від верхньої частини */
right: 10px; /* Відступ від правої частини */
bottom: 10px; /* Відступ від нижньої частини */
left: 10px; /* Відступ від лівої частини */

/* Позиція селектора */
position: absolute; /* Елемент позиціонується відносно
                    найближчого предка з non-static */
position: fixed; /* Елемент залишається на одному місці
                  при прокрутці */
position: relative; /* Елемент позиціонується відносно
                     його звичайного місця */
position: static; /* Елемент розміщується в стандартному
                   потоці документа */
position: sticky; /* Елемент позиціонується відносно
                   прокрутки */
position: inherit; /* Елемент успадковує позицію від
                   батьківського елемента */
position: initial; /* Елемент має значення за замовчуванням
                   (static) */
position: unset; /* Скидання до значення, яке залежить від
                  контексту */

/* Продовження коду у наступній сторінці */
```

```
/* Основи роботи з фоном */

/* Задає колір тексту */
background: red; /* Там можна обрати
                  зображення або кольор */

/* Колір фону */
background-color: blue;

/* Фонове зображення */
background-image: url('image.png');

/* Спосіб повторення фонового зображення */
background-repeat: no-repeat; /* Без повторення */
background-repeat: inherit; /* Спадкувати */
background-repeat: initial; /* Ініціалізує */
background-repeat: repeat; /* Повторюється по всій площі */
background-repeat: repeat-x; /* Повторюється тільки по
                              горизонталі */
background-repeat: repeat-y; /* Повторюється тільки по
                              вертикалі */
background-repeat: round; /* Розтягується, щоб поміститися
                           в контейнер */
background-repeat: space; /* Розміщується з пробілами між
                           повтореннями */
background-repeat: unset; /* Скидання */

/* Позиція фонового зображення */
background-position: top left; /* Встановлює позицію
                               зображення в верхній лівий кут */
background-position: top center; /* Позиціонує зображення в
                                   верхній центр */
background-position: top right; /* Позиціонує зображення в
                                  верхній правий кут */
background-position: center left; /* Позиціонує зображення в
                                   центрі зліва */
background-position: center center; /* Центрує зображення в
                                     середині елемента */
background-position: center right; /* Позиціонує зображення в
                                     центрі справа */
background-position: bottom left; /* Позиціонує зображення в
                                   нижній лівий кут */
background-position: bottom center; /* Позиціонує зображення в
                                     нижній центр */
background-position: bottom right; /* Позиціонує зображення в
                                    нижній правий кут */

/* Продовження коду у наступній сторінці */
```

```
background-position: 10px 20px; /* Позиціонує зображення на
                                відстані 10px зліва і 20px
                                зверху */
/* Значення по замовчуванню: top left */

/* Розмір фонового зображення */
background-size: auto;
background-size: cover;
background-size: contain;
background-size: 100px 200px; /* Встановлює ширину і
                                висоту зображення.
                                100px - ширина,
                                200px висота */
background-size: 50% 40%; /* Встановлює ширину і висоту
                            як відсоток від розміру
                            елемента */

/* Основи роботи з текстом */

/* Горизонтальне вирівнювання тексту */
text-align: left; /* Вирівнює текст по лівому краю */
text-align: center; /* Вирівнює текст по центру */
text-align: right; /* Вирівнює текст по правому краю */
text-align: justify; /* Вирівнює текст по обидва краї,
                     створюючи рівномірний інтервал */

/* Тінь тексту */
text-shadow: none; /* Відсутність тіні */

/* Розмір тіні (X, Y, Blur, Color)
X - Росташування від лівої частини селектору
Y - Росташування від верхньої частини селектору
Blur - Розмиття
Color - Кольор тіні*/
text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);

/* Обрізання тексту, що виходить за межі елемента */
text-overflow: clip; /* Обрізає текст, що виходить за
                     межі контейнера */
text-overflow: ellipsis; /* Відображає три крапки "..."
                          в кінці переповненого тексту */

/* Як обробляються пробіли та переноси рядків */
white-space: normal; /* Обрізає пробіли і переноси
                     рядків за замовчуванням */
white-space: nowrap; /* Не дозволяє переносити рядки,
                     текст продовжується в один рядок */
/* Продовження коду у наступній сторінці */
```

```
white-space: pre; /* Зберігає всі пробіли і переноси рядків */

white-space: pre-wrap; /* Зберігає пробіли,
                        але дозволяє переноси рядків */
white-space: pre-line; /* Зберігає пробіли, але видаляє зайві
                        пробіли і зберігає переноси рядків */

/* Основи роботи з коробковою моделлю */

/* Алгоритм розрахунку ширини та висоти елемента */
box-sizing: content-box; /* Розраховує ширину та висоту
                        лише для контенту */
box-sizing: border-box; /* Включає ширину і висоту
                        кордонів і відступів у
                        загальний розмір */

/* Поведінка переповненого елемента
    (visible, hidden, scroll, auto) */
overflow: visible; /* Переповнений контент видимий
                    за межами елемента */
overflow: hidden; /* Переповнений контент обрізається
                    і не видно */
overflow: scroll; /* Додає прокрутку, якщо контент
                    виходить за межі елемента */
overflow: auto; /* Додає прокрутку тільки коли
                  це необхідно */

/* Використання градієнту */
/* linear-gradient(to XXX, від_кольор, до_кольор)
   XXX - напрямок, куди повинен йти градієнт:
   right - вправо;
   left - вліво;
   top - вгору;
   bottom - вниз;
   right - вправо;
   right - вправо; */
background: linear-gradient(to top, red, yellow);
background: linear-gradient(yellow, red, yellow, blue, green);
/*
to top - напрямок (Напрямки завжди стоять на початку,
а далі кольора)

yellow, red, yellow, blue, green - набір кольорів,
які повинні бути у градієнті */
}
```

## Урок 2.5. Display-опція.

Ось коди с загальні опції у CSS:

```
селектор {  
    display: block; /* Стоїть у контейнері та  
                    ширина селектора залежна від  
                    ширини контейнера у якому  
                    знаходиться цей селектор */  
}
```

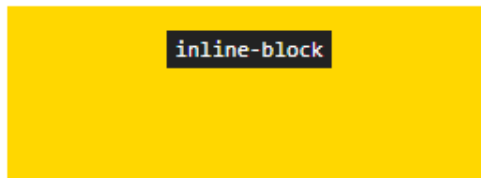
У браузері буде:



Наступний код:

```
селектор {  
    display: inline-block; /* Стоїть у контейнері та  
                           ширина селектора залежна від  
                           ширини селектора */  
}
```

У браузері буде:



Наступний код:

```
селектор {  
    display: none; /* Селектор буде прихований */  
}
```

У браузері буде:



Властивість **display** може отримувати 20 значень:

`block` - Елемент показується як блоковий. Вказавши це значення для елемента `<span>` - він почне вести себе як блоковий, тобто, його вміст буде завжди починатися з нового рядка.

`inline` - Елемент відображається як рядковий (`<span>`). Значення `inline` скасовує особливість блокових елементів завжди починатися з нового рядка.

`inline-block` - Це значення створює блоковий елемент, який поводить себе подібно рядковому елементу. Його внутрішня частина форматується як у блокового елемента, а сам елемент - як рядковий. Так себе поводить елемент `<img>`.

`inline-table` - Визначає, що елемент є таблицею `<table>`, але при цьому таблиця поводить себе як рядковий елемент і обтікається іншими елементами.

`inline-flex` - Елемент поводить себе як рядковий і викладає вміст згідно флекс-моделі. З'явився тільки у CSS3.

`flex` - Елемент поводить себе як блоковий і викладає вміст згідно флекс-моделі

`list-item` - Елемент поводить себе так само як елемент `<li>`.

`none` - Приховує елемент. Макет формується, немов елемента і не було. Зробити видимим елемент можна за допомогою скриптів. При цьому відбувається переформатування даних на сторінці з урахуванням знову доданого елемента.

`run-in` - Встановлює елемент як блоковий або рядковий залежно від контексту.

`table` - Надає елементу якостей таблиці `<table>`.

`table-caption` - Задає заголовок таблиці подібно застосуванню `<caption>`

`table-cell` - Вказує, що елемент являє собою елемент таблиці (`<td>` або `<th>`)

`table-column-group` - Елемент поводить себе ніби він елемент `<colgroup>`.

`table-column` - Елемент поводить себе ніби він елемент `<col>`.

`table-footer-group` - Використовується для зберігання однієї або декількох рядків комірок, які відображаються в самому низу таблиці. За своєю дією схожий з роботою `<tfoot>`

`table-header-group` - Елемент призначений для зберігання одного чи кількох рядків комірок, які представлені у верхній частині сторінки. За своєю дією схожий з роботою `<thead>`

`table-row` - Елемент відображається як рядок таблиці `<tr>`



`table-row-group` - Елемент аналогічний дії елемента `<tbody>`.

[initial](#) - Встановлює властивість у значення без задання

[inherit](#) - Вказує на спадковість властивості від свого батьківського елемента

**Значення без задання:** *inline*

**Наслідуює:** Ні

**Анімується:** Так

**JavaScript синтаксис:** `object.style.display="none"`

## Урок 2.6. Фільтри.

CSS властивість `filter` визначає, які візуальні ефекти застосувати до елемента, наприклад, розмиття чи контраст. Найчастіше застосовується до елемента `<img>`).

Властивість **filter** може отримувати 14 значень:

`none` - Ефекти відсутні. Без задання.

`blur(px)` - Задає розмиття. Чим більше значення, тим більше розмиття.

Якщо значення не вказано, використовується 0. Від'ємне значення заборонено.

`brightness (%)` - Яскравість зображення:

- 0% зробить зображення повністю чорним.
- 100% (1) Зображення залишиться без змін. Без задання.
- Значення більше за 100% зробить зображення більш світлим.

`contrast (%)` - Контрастність зображення:

- Значення менше 100% чи 1 зменшить контрастність.
- 100% чи 1 Зображення залишиться без змін. Без задання.
- Значення більше 100% чи 1 збільшить контрастність.

Від'ємне значення не допускається. Пусте значення сприймається як 1.

`grayscale (%)` - Перетворить зображення у чорно-біле.

- 100% чи 1 - Перетворює зображення в чорно-біле.
- 0% чи 0- Зображення не змінюється.
- Від 0% до 100% (або від 0 до 1) змінюють кольорову гамму картинки.

Від'ємні значення заборонені. Порожнє значення сприймається як 0.

`hue-rotate (deg)` - Змінює кольоровість зображення за рахунок повороту відтінку на колірному колі.

- 0 або 360 градусів залишає зображення незмінним.
- Будь-які інші значення будуть змінювати колірну гамму.

Від'ємні значення дозволені. Максимальне значення 360deg. Порожнє значення сприймається як 0deg.

`invert (%)` - Інвертує кольору в зображенні. За своєю дією схоже на перетворення фотографії в негатив.

- 0% або 0 - Зображення буде без змін.
- 100% або 1 - Повністю інвертує кольору картинки.
- Від 0% до 100% або від 0 до 1 - Частково інвертує кольори.

Від'ємні значення заборонені. Порожнє значення сприймається як 0.

`opacity (%)` - Задає ступінь прозорості зображень. За своєю дією схожа на властивість `opacity`, але деякі браузері для фільтрів застосовують апаратне прискорення щоб підвищити їх продуктивність.

- 100% або 1 Зображення залишиться без змін.
- 0% або 0 Зображення стане повністю прозорим.
- 0% до 100% або від 0 до 1 задає ступінь прозорості зображення.

Від'ємне значення не допускається. Порожнє значення сприймається як 1.

`saturate (%)` - Насиченість кольорів в зображенні.

- 0% або 0 прибирає насиченість кольорів в зображенні, перетворюючи його в чорно-біле.
- 100% або 1 залишає зображення незмінним.
- 100% та більше робить зображення більш насиченими.

Від'ємне значення не допускається. Порожнє значення сприймається як 1.

`sepia (%)` - Перетворює зображення в сепію - так зване чорно-біле зображення з коричневим відтінком. Сепія надає фотографіям старовинний вигляд:

- 0% або 0 Зображення не змінюється.
- 100% або 1 Сепія в максимальну силу.
- 0% до 100% або від 0 до 1 лінійно застосовують сепію.

Від'ємні значення не допускаються. Порожнє значення сприймається як 0.

`url ()` - Приймає розташування XML файлу, який визначає SVG фільтр. Посилання, також може включати в себе якір для конкретного фільтра.

Приклад: `filter: url(svg-url#element-id)`

[initial](#) - Встановлює властивість у значення без задання

[inherit](#) - Вказує на спадковість властивості від свого батьківського елемента

`drop-shadow(h-shadow v-shadow blur spread color)`

Застосовується ефект тіні до зображення.

Можливі значення:

- `h-shadow` - Обов'язково. Задає значення ширини горизонтальної тіні. Від'ємні значення відобразять тінь зліва від зображення.
- `v-shadow` - Обов'язково. Задає значення пікселя для вертикальної тіні. Від'ємні значення відобразять тінь над зображенням.
- `blur` - Необов'язково. Це третє значення і воно має бути в пікселях. Додає ефект розмиття тіні. Більше значення буде створювати більше розмиття (тінь стає більшою та світлішою). Від'ємні значення не допускаються. Якщо значення не вказано, використовується 0 (тіні не буде видно).
- `spread` - Необов'язково. Це четверте значення і воно має бути в пікселях. Додатнє значення змусить тінь розширюватися, а від'ємні значення змусять тінь скорочуватися. Якщо це значення не вказано, то воно буде 0 (тінь матиме той же розмір, що й елемент). Примітка: Chrome, Safari, Opera можливо й інші браузері, не підтримують це значення.

- колір - необов'язково. Додає колір тіні. Якщо не вказано, то колір залежить від браузера (найчастіше, тінь буде чорного кольору).

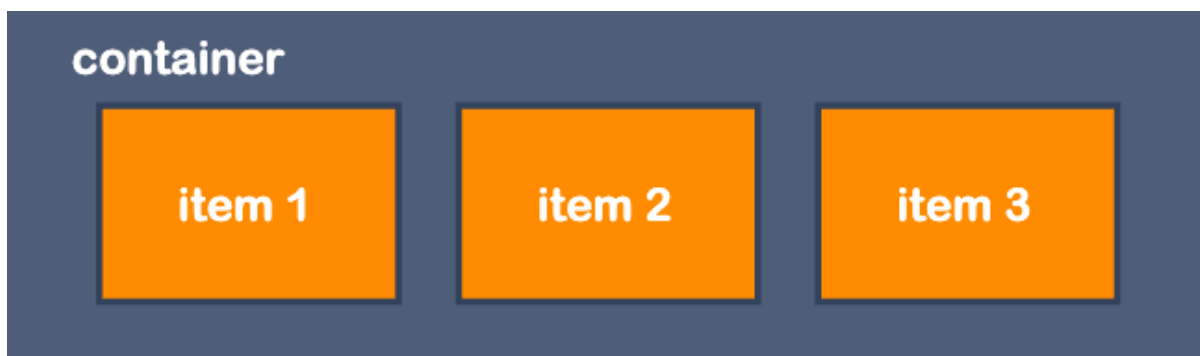
Приклад створення червоної тіні, ширина та висота якої 8px з ефектом розмивання в 10 пікселів:

```
filter: drop-shadow(8px 8px 10px red);
```

Підказка: Цей фільтр аналогічний властивості `box-shadow`.

## Урок 2.7. Flexbox.

**Flexbox**, скорочено від **Flexible Box Layout**, - це потужний модуль верстки в CSS, призначений для оптимізації вирівнювання, розподілу та організації елементів всередині контейнера.



Модель компоновання **flexbox** працює на основі зв'язку "батько-дитина". Батько, який називається **флекс-контейнер**, відповідає за диктування макета своїх дочірніх елементів, відомих як **флекс-елементи**. Використовуючи ряд властивостей flexbox, ми можемо точно контролювати розмір, позицію і порядок розташування флекс-елементів у флекс-контейнері.

Примітка! Усередині флекс-контейнера багато звичайних правил позиціонування не застосовуються так, як очікувалося. Давайте розглянемо наступний список:

- Робота з флекс-елементами відрізняється від роботи з елементами **inline** або **block-level**.
- На відміну від елементів **block-level**, флекс-елементи не перекривають інші елементи і не накладаються один на одного.
- Поля, застосовані на краях батьківського елемента, не виходять за його межі.

## flex модель

Оскільки ми не маємо справу з вбудованими або блоковими елементами, потік документів у флекс-моделі визначається осями флекс-контейнера, за якими вирівнюються всі елементи. Щоб ініціювати поведінку flexbox, ми застосовуємо властивість `display: flex;` до батьківського елемента, що містить усі елементи.

Ми можемо впливати на напрямок розташування flex-елементів. За замовчуванням, flex-елементи розташовуються у ряд. Однак ми можемо змінити цю поведінку за допомогою властивості `flex-direction`. Давайте порівняємо документообіг без флексу та з флексом:

Код на CSS:

```
/* Apply flex to the parent element.
/* In this situation, it is the <ul> element with the
"list" class name.
*/
.list {
  display: flex;
}

/* Add border to the items of the flex container */
.list li {
  border: 1px solid forestgreen;
}

/* Remove default decoration */
ul {
  list-style: none;
}
```

Код на HTML:

```
<html lang="en">
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <!-- Default document flow -->
    <h4>Default</h4>
    <ul>
      <li>Lorem ipsum dolor sit amet.</li>
      <!-- Продовження коду у наступній сторінці -->
```

```

    <li>Lorem ipsum dolor sit amet consectetur.</li>
    <li>Lorem ipsum dolor sit amet.</li>
  </ul>

  <!-- Applied flexbox -->
  <h4>Flex</h4>
  <ul class="list">
    <li>Lorem ipsum dolor sit amet.</li>
    <li>Lorem ipsum dolor sit amet consectetur.</li>
    <li>Lorem ipsum dolor sit amet.</li>
  </ul>
</body>
</html>

```

Результат на сторінці:

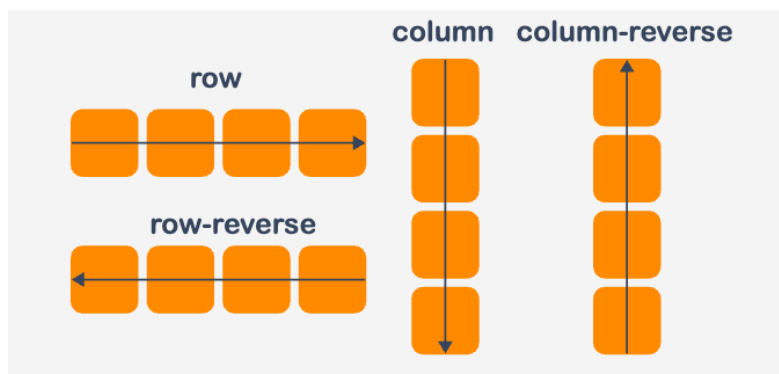
### Default

- Lorem ipsum dolor sit amet.
- Lorem ipsum dolor sit amet consectetur.
- Lorem ipsum dolor sit amet.

### Flex

Властивість `flex-direction` дозволяє нам визначити напрямок, в якому розташовані `flex`-елементи. За замовчуванням ця властивість має значення `row`, що призводить до компонування зліва направо у браузерях, які використовують англійську мову за замовчуванням. Ось значення, які можна вставити встановити значення: `flex-direction: row | column | row-reverse | column-reverse;`

Ось як розташовуються елементи з використанням `flex-direction`:



Розглянемо для прикладу наступну навігацію по сайту

Ось код HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <ul class="navigation-list">
      <li class="navigation-item">
        <a href="#" class="navigation-link">Home</a>
      </li>
      <li class="navigation-item">
        <a href="#" class="navigation-link">About</a>
      </li>
      <li class="navigation-item">
        <a href="#" class="navigation-link">Price</a>
      </li>
      <li class="navigation-item">
        <a href="#" class="navigation-link">Team</a>
      </li>
      <li class="navigation-item">
        <a href="#" class="navigation-link">Support</a>
      </li>
    </ul>
  </body>
</html>
```

А це CSS-код:

```
.navigation-list {
  background-color: honeydew;
  display: flex;
  padding: 10px;
  flex-direction: XXX;
}

.navigation-item {
  padding: 10px;
  border-radius: 6px;
  background-color: violet;
} /* Продовження коду у наступній сторінці */
```

```

.navigation-item:not(:last-child) {
  margin-right: 20px;
}

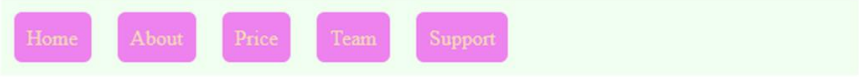

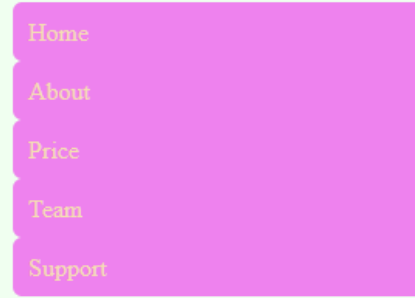

.navigation-link {
  color: wheat;
}

ul {
  list-style: none;
}

a {
  text-decoration: none;
}

```

Де було XXX, там будемо указувати значення. Ось таблиця, коли XXX дорівнює якомусь значенню:

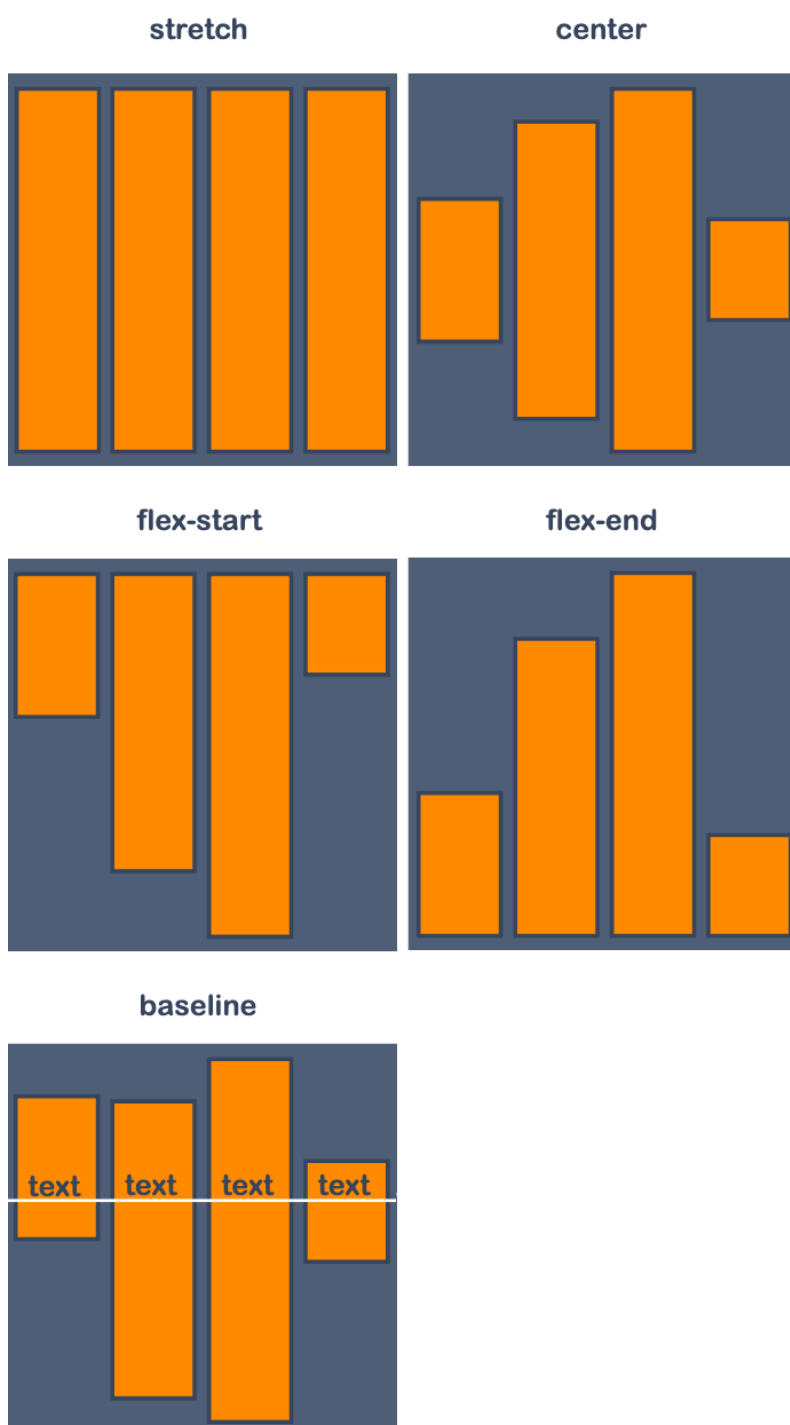
Коли XXX дорівнює:	Результат
row	
row-reverse	
column	
column-reverse	



## align-items

Властивість align-items в CSS використовується для вирівнювання елементів по вертикалі в контейнері з display: flex або display: grid. Для таблиць ця властивість також не застосовується безпосередньо, але ми можемо використовувати Flexbox або Grid всередині комірок таблиці для досягнення потрібного вирівнювання: align-items: stretch | center | flex-start | flex-end | baseline;

Ось як розташовуються елементи з використанням align-items:



Тепер зробимо коди:

HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul class="cards-list">
      <li class="card">
        
        <p>Lorem, ipsum dolor.</p>
      </li>
      <li class="card">
        
        <p>
          Lorem ipsum dolor sit, amet consectetur.
        </p>
      </li>
      <li class="card">
        
        <p>Lorem ipsum dolor sit amet.</p>
      </li>
      <li class="card">
        
        <p>
          Lorem ipsum dolor sit, amet consectetur.
        </p>
      </li>
      <li class="card">
        
        <p>
          Lorem ipsum dolor sit, amet consectetur.
        </p>
      </li>
    </ul>
  </body>
</html>
```

CSS:











```
.cards-list {
  display: flex;
  border: 1px solid khaki;
  align-items: XXX;
  width: 660px;
}
















.card {
  border: 1px dashed navy;
  padding: 10px;
  width: 120px;
}

.card img {
  display: block;
  margin: 0 auto;
}

/* Remove default decorations */
ul {
  list-style: none;
  padding-left: 0;
}
```

Де було XXX, там будемо указувати значення. Ось таблиця, коли XXX дорівнює якомусь значенню:

Коли XXX дорівнює:	Результат				
stretch	<div><p>Lorem, ipsum dolor.</p></div>	<div><p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p></div>	<div><p>Lorem ipsum dolor sit amet.</p></div>	<div><p>Lorem ipsum dolor sit, amet consectetur adipisicing elit.</p></div>	<div><p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p></div>
center	<div><p>Lorem, ipsum dolor.</p></div>	<div><p>Lorem ipsum dolor sit, amet consectetur.</p></div>	<div><p>Lorem ipsum dolor sit amet.</p></div>	<div><p>Lorem ipsum dolor sit, amet consectetur.</p></div>	<div><p>Lorem ipsum dolor sit, amet consectetur.</p></div>

flex-start	<div>  <p>Lorem, ipsum dolor.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div> <div>  <p>Lorem ipsum dolor sit amet.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div>
flex-end	<div>  <p>Lorem, ipsum dolor.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div> <div>  <p>Lorem ipsum dolor sit amet.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div>
baseline	<div>  <p>Lorem, ipsum dolor.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div> <div>  <p>Lorem ipsum dolor sit amet.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div> <div>  <p>Lorem ipsum dolor sit, amet consectetur.</p> </div>

## Урок 2.8. Анімація.

Іноді нам доводиться створювати анімацію для зміни елементів на веб-сторінці з контролюючими факторами, такими як швидкість, час затримки та тривалість. У таких випадках для виконання цього завдання може бути використана властивість "transition". Елемент завжди має два стани: **початковий** і **кінцевий**. Ми можемо керувати поведінкою зміни елемента за допомогою таких властивостей:

```
.transitionClass {
  /* Ім'я властивості CSS, яку нам
  потрібно анімувати (Значення повинно
  мати назву тієї опції, яка повинна
  змінитись при зміні статусу
  селектора) */
  transition-property: color;

  /* Час, протягом якого нам потрібно
  змінити стан елемента. Він вказується
  в секундах (s) або мілісекундах (ms) */
  transition-duration: 300ms;
  /* Продовження коду у наступній сторінці */
}
```

```

/* Він задає криву швидкості ефекту
переходу. Типовими значеннями є:
ease; linear; ease-in; ease-out; ease-in-out.*/
transition-timing-function: ease;

/* Затримка часу до початку ефекту переходу.
Він вказується в секундах (s) або
мілісекундах (ms). */
transition-delay: 2s;

/* 1 секунда = 1000 мілісекунда */
}

```

Ось приклад сторінки, у якому є контейнер (Тег <div>), при наближення курсору мишки до нього зміниться його колір:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <style>
      div {
        width: 400px;
        height: 100px;
        border-radius: 10px;
        border: 2px solid orange;
        background-color: yellow;
        transition-property: background-color;
        transition-duration: 300ms;
        transition-timing-function: ease-in-out;
        transition-delay: 100ms;
      }

      div:hover {
        background-color: darkblue;
      }
    </style>
  </head>
  <body>
    <div></div>
  </body>
</html>

```

Якщо треба змінювати значення усіх опцій, треба тоді вказувати:

```
transition-property: all;
```

У CSS властивість `transition-timing-function` визначає, як швидкість анімації змінюється протягом часу. Ось пояснення основних значень:

**1. ease:**

- Це значення за замовчуванням.
- Швидкість анімації спочатку зростає, потім сповільнюється.
- Створює ефект прискорення на початку та сповільнення в кінці.
- Графік швидкості має S-подібну форму.

**2. linear:**

- Швидкість анімації є постійною протягом усього часу.
- Немає прискорення або сповільнення.
- Графік швидкості являє собою пряму лінію.

**3. ease-in:**

- Швидкість анімації зростає на початку, потім досягає постійного значення.
- Створює ефект прискорення на початку.
- Графік швидкості починається з нуля і прискорюється.

**4. ease-out:**

- Швидкість анімації спочатку є постійною, потім сповільнюється до кінця.
- Створює ефект сповільнення в кінці.
- Графік швидкості починається зі швидкості і сповільнюється до кінця.

**5. ease-in-out:**

- Швидкість анімації спочатку зростає, потім сповільнюється.
- Створює ефект прискорення на початку і сповільнення в кінці.
- Графік швидкості має S-подібну форму, але більш виражений, ніж у `ease`.

Також можна писати просто transition у таблиці стилів:

```
transition: all 1200ms ease-in-out 250ms;  
/* 1200ms - the time duration; 250ms - the delay */
```

Є також анімація, яка дає команду селектору крутитися за часовою стрілкою:

```
/* Зробити це 5 раз */  
animation-iteration-count: 5;  
  
/* Зробити це безкінечно */  
animation-iteration-count: infinite;
```

Також є опція animation-direction, яка застосовується таким чином:

```
/* елемент переміщується зліва направо,  
повторюється знову з початку. */  
animation-direction: normal;  
  
/* елемент переміщується справа наліво,  
повторюється знову з кінця */  
animation-direction: reverse;  
  
/* елемент переміщується зліва направо,  
потім справа наліво, і так далі */  
animation-direction: alternate;  
  
/* елемент переміщується справа наліво,  
потім зліва направо, і так далі */  
animation-direction: alternate-reverse;
```

## Урок 2.9. Адаптація.

Адаптація сайту може включати різні аспекти, такі як дизайн, функціональність, продуктивність та відповідність сучасним стандартам. Адаптивний дизайн забезпечує коректне відображення сайту на різних пристроях, від мобільних телефонів до настільних комп'ютерів. Оптимізація швидкості завантаження сайту включає використання кешування, зменшення розміру зображень, мінімізацію CSS та JavaScript файлів. Забезпечення доступності сайту для людей з обмеженими можливостями, включає використання відповідних атрибутів ARIA, забезпечення високої контрастності тексту та підтримка навігації з клавіатури. Адаптація сайту для підтримки різних мов включає використання фреймворків для локалізації та перекладу контенту.

Ось синтаксис при використанні адаптації:

```
@media <media_type> <operator> (<media_feature>) {  
  /* CSS rules */  
}
```

1. <media\_type> - декларує тип пристрою. Можливі значення:
  - all - значення за замовчуванням. Якщо нічого не вказано, правило носія працює для всіх пристроїв;
  - print - правило носія працює для пристроїв, які виробляють друковані документи, наприклад, принтерів;
  - screen - правило носія працює для пристроїв, які виробляють друковані документи, наприклад, принтерів.
2. <media\_feature> - заявляє характеристики пристрою. Найбільш поширене застосування:
  - min-width: - мінімальна ширина вікна перегляду;
  - max-width: - максимальна ширина вікна перегляду.
3. <operator> - media-type та media-feature відокремлюються необов'язковим логічним оператором. Його значеннями можуть бути: and або or,.

```
/* 1-ий приклад */  
@media screen and (max-width: 1200px) {  
  .container {  
    color: aliceblue;  
  }  
}  
  
/* 2-ий приклад */  
@media screen and (min-width: 1680px) {  
  .container {  
    color: gainsboro;  
  }  
}  
  
/* У першому прикладі ми сказали браузеру,  
що для будь-яких пристроїв із шириною екрана  
менше 1200 пікселів або дорівнює 1200 пікселів  
застосувати властивість color зі значенням  
aliceblue до елемента з контейнером імені класу.  
(Продовження коду у наступній сторінці)
```



У другому прикладі ми повідомили браузеру, що якщо ширина екрана будь-якого пристрою перевищує 1680 пікселів або дорівнює 1680 пікселів, властивість кольору зі значенням gainsboro має бути застосовано до елемента з назвою класу контейнера. \*/

## III РОЗДІЛ. СКРИПТИ.

### Урок 3.1. Що таке JavaScript.

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

Якщо ми вивчали мови HTML та CSS, які не відносяться до мов програмування, то JavaScript відноситься до мов програмування, а HTML та CSS відносяться до мов, які будують графіку, бо вони не можуть виконувати цикли та на цих мовах не можливо зробити функції, яке може зробити лише мова програмування, такі як JavaScript.

### Урок 3.2. Додавання та виконання скриптів на веб-сторінці.

Щоб додати JS-скрипти, можна зробити 3 способами:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- 1 спосіб - додавання тегу <script> -->
    <script>
      console.log("Hello World");
    </script>

    <!-- 2 спосіб - додавання посилання на
    скрипт до тегу <script> -->
    <script src="шлях/до/скрипта.js"></script>
    <!--Продовження коду є у наступній сторінці -->
```

```

        <!-- 3 спосіб - додавання посилення на
        скрипт до тегу <script> з інтернету -->
        <script
            type="application/javascript"
            src="http://сторінка.зі.скриптом/"
        ></script>
    </body>
</html>

```

### Урок 3.3. Події.

Події – атрибути у тегах, які виконують скрипт при зміні стану тегу. Ось код з базовими атрибутами подій, які треба знати:

```

<!-- При натискання на тег -->
<a onclick="console.log('Hello World');"></a>

<!-- Викликається після завантаження елемента завершено -->
<a onload="console.log('Hello World');"></a>

<!-- Викликається, коли сторінка розвантажена, або вікно
браузера було зачинено. -->
<a onunload="console.log('Hello World');"></a>

<!-- Скрипт виконується, коли елемент втрачає фокус. -->
<a onblur="console.log('Hello World');"></a>

<!-- Викликається в той момент, коли елемент отримує фокус. -->
<a onfocus="console.log('Hello World');"></a>

<!-- Викликається, коли користувач щось пише в поле пошуку (для
<input type="search">) -->
<a onsearch="console.log('Hello World');"></a>

<!-- Викликається після того як будь-який текст був обраний в
елементі. -->
<a onselect="console.log('Hello World');"></a>

<!-- Викликається при відправленні форми -->
<a onsubmit="console.log('Hello World');"></a>

<!-- Подія викликається, коли користувач затис
(натиснув та не відпускає) клавішу. -->
<a onkeydown="console.log('Hello World');"></a>
<!--Продовження коду є у наступній сторінці -->

```

```
<!-- Викликається коли користувач натиснув на клавішу -->
<a onkeypress="console.log('Hello World');"></a>

<!-- Викликається коли користувач відпускає клавішу -->
<a onkeyup="console.log('Hello World');"></a>

<!-- Виникає при подвійному клацанні лівою кнопкою
мишки на елементі -->
<a ondblclick="console.log('Hello World');"></a>

<!-- Викликається, коли користувач затискає лівою кнопкою мишки
на елементі. -->
<a onmousedown="console.log('Hello World');"></a>

<!-- Викликається коли курсор миші переміщається над елементом
-->
<a onmousemove="console.log('Hello World');"></a>

<!-- Викликається, коли курсор виходить за межі елемента -->
<a onmouseout="console.log('Hello World');"></a>

<!-- Виконується, коли курсор наводиться на елемент -->
<a onmouseover="console.log('Hello World');"></a>

<!-- Викликається, коли користувач відпускає кнопку миші -->
<a onmouseup="console.log('Hello World');"></a>

<!-- Викликається, коли користувач копіює вміст елемента -->
<a oncopy="console.log('Hello World');"></a>

<!-- Викликається, коли користувач вирізає вміст елемента -->
<a oncut="console.log('Hello World');"></a>

<!-- Викликається, коли користувач вставляє вміст в елемент -->
<a onpaste="console.log('Hello World');"></a>

<!-- Виконується при перериванні якоїсь події -->
<a onabort="console.log('Hello World');"></a>
```

Це ті події, які треба знати. Вони працюють на усіх версіях HTML. Але є події у HTML, які працюють лише на HTML 5.0. Таблиця з подіями, які працюють лише у HTML 5.0 є на наступній сторінці.

Події	Стан тегу
onplay	Скрипт викликається коли медіа дані готові почати відтворення.
onafterprint	Скрипт виконується тільки після як документ надрукований.
onbeforeprint	Скрипт виконується перед тим, як документ надрукований.
onbeforeunload	Скрипт виконується коли документ ось-ось буде вивантажений
onhashchange	Скрипт виконується коли там відбулися зміни до частини якоря в URL
onmessage	Скрипт виконується коли викликане повідомлення.
onoffline	Спрацьовує коли браузер починає працювати в автономному режимі
ononline	Спрацьовує коли браузер починає працювати в режимі онлайн.
onpagehide	Скрипт виконується коли користувач переходить на іншу сторінку сторінку.
onpageshow	Скрипт виконується коли користувач заходить на сторінку.
onpopstate	Скрипт виконується коли змінено історію одного вікна.
onresize	Скрипт виконується, коли розмір вікна браузера змінюється.
onstorage	Скрипт виконується, коли вміст Web Storage оновлюється.
oncontextmenu	Скрипт виконується коли викликається контекстне меню.
oninput	Скрипт викликається коли користувач вводить дані поле.
oninvalid	Скрипт виконується, коли елемент недійсний.
onreset	Викликається, коли натискається у формі кнопка типу Reset.
ondrag	Періодично викликається при операції перетягування.
ondragend	Викликається коли користувач відпускає перелягуваний елемент.
ondragenter	Викликається, коли перетягуваний елемент входить в цільову зону.

ondragleave	Викликається, коли перетягуваний елемент виходить з зони призначення.
ondragover	Викликається, коли перетягуваний елемент знаходиться в зоні призначення.
ondragstart	Викликається, коли користувач починає перетягувати елемент, або виділений текст.
ondrop	Викликається, коли перетягуваний елемент падає до зони призначення.
onscroll	Викликається при прокручуванні вмісту елемента (чи веб-сторінки).
onwheel	Викликається, коли користувач прокручує коліщатко миші.
oncanplay	Скрипт виконується коли файл готовий, для початку відтворення (коли він буферизований достатньо, щоб почати відтворення)
oncanplaythrough	Скрипт виконується, коли контент вже може бути відтворений без переривання на буферизацію.
oncuechange	Скрипт виконується коли змінюється кий в <track> елемента
ondurationchange	Викликається коли змінюється довжина медіа файлу.
onemptied	Викликається коли доступ до медіа контенту обривається (зникло з'єднання з мережею).
onended	Викликається коли медіа елемент повністю відтворив свій зміст.
onshow	Викликається, коли елемент <menu> буде відображено як контекстне меню.
onloadedmetadata	Скрипт виконується коли метадані (розміри чи тривалість) завантажуються.
onloadeddata	Викликається коли медіа данні завантажено.
onloadstart	Викликається коли браузер тільки починає завантажувати медіа дані з сервера.
onpause	Викликається коли відтворення медіа даних призупинено.
onplaying	Викликається коли розпочато відтворення медіа даних.
onprogress	Подія onprogress відбувається, коли браузер завантажує вказане аудіо / відео.
onratechange	Викликається коли змінюється швидкість відтворення медіа даних.
onseeked	Викликається коли атрибут seeked у тега audio або video змінює значення з true на false.

onseeking	Викликається коли атрибут seeking у тегів audio або video змінює значення з false на true
onstalled	Скрипт виконується коли браузер з будь-якої причини не може отримати медіа дані.
onsuspend	Скрипт виконується коли з будь-якої причини завантаження даних призупинено до його повного завантаження.
ontimeupdate	Викликається коли змінилася позиція відтворення елемента <audio> або <video>.
onvolumechange	Викликається коли змінюється гучність звуку.
onwaiting	Викликається коли наступний кадр при відтворенні медіа даних недоступний, але браузер очікує що він незабаром завантажиться.
ontoggle	Викликається, коли користувач відкриває або закриває елемент <details>.
onerror	Викликається якщо при завантаженні елемента сталася помилка.

### Урок 3.4. Перші скрипти.

Де згадувалося про команду: «`console.log("Hello World");`», ця команда служить для відображення значення у консолі. Ось усі команди, які можна застосовувати:

```
console.log("Hello World"); // Відображення значення у консолі
/*
Де "Hello World" - це значення.
Де ; - це означає, що команда закінчена. Знак ";"
обов'язково
треба ставити після того, коли було написано команда.
Де "/" або "/* текст */" - це коментарі.
"/" - коментар лише на одну строку, а "/* текст */" -
коментар на декілька строків.
*/

alert("Hello World"); /* Відображення повідомлення у браузері
*/
window.alert("Hello World"); /* Теж саме, як "alert("Hello World");"
return confirm("Hello World"); /* Відображення повідомлення у
браузері з кнопками "Так" чи "Ні" */
```

У JavaScript, як і усіх мовах програмування, є змінні. Змінні служать для збереження значень у себе. Зберігати значення можна у **константі**, які

ніколи не змінюються, або у **змінній**, яка може змінюватись. Ось приклад створення змінних та констант:

```
const cnst = "Hello World"; // Створення константи
let var = "Hello World"; // Створення змінної
/*
const, var та let - оголошення про створення змінних.
"Hello World" - значення змінних.
cnst та var - назва змінних
Назва змінних може бути будь якою, але не може
називатись службовими словами, такі як: let,
class, return і function. Неанглійські літери у назв
змінних дозволені, але не рекомендуються.
Змінні з іменами "apple" і "APPLE" - це дві різні
змінні, бо регістр має значення. Також змінні не
можуть мати знаки (Такі як: ., ., ?)
*/

// Можна робити таким чином:
let var2;
var2 = "Hi";

// Таким чином можна і змінювати значення змінних
let var3 = "Hello"; // Створення нової змінної та
                    // задання значення для нової
                    // змінної
var3 = "Hi"; // Зміна значення змінної
```

Усі значення змінних цього коду – текстові. JavaScript має такі типи значень:

Тип значення	Приклад
string (текстові)	“Hi”; “Text”; “Hello World”
number (числові)	5; 6.24; 100.00; 3.6; 9; 10
boolean (Істина)	<b>true</b> або <b>false</b>
array (Списки)	[3, “Hi”, “World”, false]

Ось код зі змінними різних типів значень:

```
let str = "Hello";
let num = 5.4;
let bool = true;
let arr = ["Hi", ["Hello World", 4, true], 3, false];

// Продовження коду у наступній сторінці
```



```
/* array (Список) - тип значення, який може вмістити у себе декілька
значень (У тому числі і змінних, бо змінна працює як посилання на
значення). Якщо зробимо список з змінних, які є у цьому коді, такі
як: str; num; bool; arr, то у коді ми напишемо як:
[str, num, bool, arr], то JavaScript буде це розуміти,
як: ["Hello", 5.4, true, ["Hi", ["Hello World", 4, true], 3, false]]
```

Щоб дізнатися, який тип значення має змінна, то можна зробити таким чином: \*/

```
let typeOfStr = typeof str; /* typeof допомагає дізнатися про тип
значення змінної.
```

Ми цього можемо не дізнатись, бо ми зберігали інформацію у змінній. Для того, щоб побачити цю інформацію, треба вивести значення цієї змінної у консолі: `*/`  
`console.log(typeofStr)`

```
/* Залишилось з нами дізнатися про змінних – це про вибору елементу з списку. Для вибору елементу з списку використовуємо таке розуміння, як індекс. Індекс – це елемент перелічуваної множини, який вказує на певний елемент масиву. Зазвичай це невід'ємне ціле число. У деяких мовах негативні індекси використовуються для підрахунку елементів у зворотному напрямку (починаючи з кінця масиву). Для обирання останнього елементу з списку, використовуємо індекс [-1], використовуємо [-2] для пред-останнього, використовуємо [-3] для пред-пред-останнього, тощо. Це що стосується обирання елементу по списку від останнього до першого. Для пошуку елементів від початку до кінця списку ми віднімаємо 1 від числа звичайного впорядкованості елементу. Наприклад: перший елемент – [0], другий елемент – [1], третій елемент – [2], тощо. Ось змінна з списком та змінні, які показують елементи за індексом: */
```

```
let arrExample = ["First", "Second", "Third", "Fourth"];
let var1 = arrExample[0]; // Зберігає значення у змінній, як: "First"
let var2 = arrExample[2]; // Зберігає значення у змінній, як: "Second"
let var3 = arrExample[-1]; // Зберігає значення у змінній, як: "Fourth"
let var4 = arrExample[-3]; // Зберігає значення у змінній, як: "Second"
arrExample[2] = "3rd"; // Змінює елемент за індексом [2] на "3rd"
let var5 = arrExample.push("First"); // Створює елемент "First"
// у кінці списку
let var8 = arrExample.unshift("Second"); // Створює елемент "Second"
// у початку списку
let var6 = arrExample.pop(); // Зберігає значення останнього елементу
// та видаляє останній елемент списку
let var7 = arrExample.shift(); // Зберігає значення першого елементу
// та видаляє перший елемент списку
```

Тепер перейдемо до діалогового повідомлення з кнопками «ОК» та «Cancel», яке з'являється у браузері:

```
let var1 = confirm("Питання"); // Створення діалогового повідомлення
                                // з кнопками "OK" та "Cancel"
alert(var1); /* Якщо користувач натиснув на "OK", то
var1 = true, якщо користувач натиснув на "Cancel", то
var1 = false */
```

Тепер розберемо використання елементів з id. Загалом класи використовують для CSS, а id для JavaScript. Але id можна використовувати у CSS та у JavaScript, так і класи можна використовувати у CSS та у JavaScript. Ось як можна отримати теги через id або через класи:

```
<!DOCTYPE html>
<html>
  <head><meta charset="UTF-8"></head>
  <body>
    <p id="blinking" class="cls">
      Text
    </p>
    <script>
      // Бере теги з id, яке вказано у функції:
      const p = document.getElementById("blinking")

      // Бере теги з class, яке вказано у функції:
      const p2 = document.getElementsByClassName("cls")

      setInterval(function () {
        if (p.style.fontSize !== "10px") {
          p.style.fontSize = "10px" // Зміна шрифту
          p.style.color = "Red" // Зміна кольору
          p.textContent = "hi" // Зміна Тексту
        } else {
          p.style.fontSize = "20px"
          p.style.color = "Yellow"
          p.textContent = "Hello"
        }
      }, XXX) /* setInterval виконує функцію або команду
                постійно та після завершення команди, чекає
                XXX мілісекунд, а потім далі робить цю
                функцію чи команду */
      p.remove(); // Видалити елемент з id="blinking"
    </script>
  </body>
</html>
```

### Урок 3.5. Оператори, умови, розгалуження та цикли.

Оператори діляться на **базові** та **логічні**. Логічні оператори зазвичай використовуються з булевими (булевими) значеннями; однак значення, яке вони повертають, також є логічним. Ось таблиця з логічними операторами:

Оператор	Перевірка
==	Дорівнює
===	Суворо дорівнює
!=	Не дорівнює
!==	Суворо не дорівнює
>	Менше
<	Більше
>=	Менше або дорівнює
<=	Більше або дорівнює
	АБО
&&	ТА
!	НІ

Ось які будуть булеві значення у змінних:

```
let var1 = 3 > 5 || 6 >= 4; // true
let var2 = 5 === "5" || 6 === "6"; // false
let var3 = 5 == "5" || 6 == "6"; // true
let var4 = 3 > 5 && 6 >= 4; // false
let var5 = !(3 > 5 || 6 >= 4); // false
```

Ось таким чином змінні можуть зберігати значення, які можуть працювати, як умови.

До речі! Як ви і замітили, що є оператор «Дорівнює» та «Суворо дорівнює». Ці дві оператори різняться тим, що оператор «Дорівнює» може порівняти значення, не звертаючи уваги на різний тип значень, то оператор «Суворо дорівнює» буде порівнювати ще і тип значення. Така схожа різниця є між операторами «Не дорівнює» та «Суворо не дорівнює».

Тепер розглянемо розгалуження на JavaScript. Розгалуження – схема алгоритмів при якому, якщо умова є істиною, то робиться за умовою, а у іншому випадку буде виконуватись інша дія. Ось як виглядають розгалуження у JavaScript:

```
// Неповне розгалуження: ЯКЩО (...) ТО {...}
if (/* Умова */) {
    // Перелік команд
}

// Продовження коду у наступній сторінці
```

```
// Повне розгалуження:
// ЯКЩО (...) ТО {...}, ІНАКШЕ {...}
if (/* Умова */) {
    // Перелік команд
} else {
    // Перелік команд
}

// Повне розгалуження:
// ЯКЩО (...) ТО {...}, ІНАКШЕ ЯКЩО (...) ТО {...}, ..., ІНАКШЕ {...}
if (/* Умова */) {
    // Перелік команд
} else if {
    // Перелік команд
} else {
    // Перелік команд
}
```

Тепер поговоримо про базові оператори. Базові оператори служать для виконання математичних дій, такі як: множення; ділення; додати; тощо. Ось використання їх у коді:

```
// При використанні текстових
var varText1 = "Hello";
var varText2 = "World";
// var - такий самий, як let

var TextSum = varText1 + varText2;
/* У змінній значення буде: "HelloWorld" */

var TextMultiply = varText1 * 3;
/* У змінній значення буде: "HelloHelloHello"
Де було вказано 3 - ми вказували число, бо
текст множити на текст неможливо. Але
текст множити на ціле додатне число можна.*/

// Що стосується числових
var varNum1 = 4;
var varNum2 = 3;

var plus = varNum1 + varNum2; // Додавання
/* У змінній значення буде: 7 */

// Продовження коду у наступній сторінці
```

```

var minus = varNum1 - varNum2; // Віднімання
/* У змінній значення буде: 1 */

var multiply = varNum1 * varNum2; // Множення
/* У змінній значення буде: 12 */

var divide = varNum1 / varNum2; // Ділення
/* У змінній значення буде: 1.3333333333333333 */

var remainder = varNum1 % varNum2; // Отримання остачі ділення
/* У змінній значення буде: 1 */

var degree = varNum1 ** varNum2; // Узвести у степінь
/* У змінній значення буде: 64 */

```

Також у JS є застосовування. Нехай у нас буде змінна *varApply*, яка зберігає у себе число 5. Ось як буде працювати застосовування:

Назва застосовування	1-ий спосіб	2-ий спосіб	Результат
Застосувати додавання (+=)	<code>varApply += 5;</code>	<code>varApply = varApply + 5;</code> 5+5	10
Застосувати віднімання (--)	<code>varApply -= 5;</code>	<code>varApply = varApply + 5;</code> 5-5	0
Застосувати множення (*=)	<code>varApply *= 5;</code>	<code>varApply = varApply + 5;</code> 5*5 або 5x5	25
Застосувати ділення (/=)	<code>varApply /= 5;</code>	<code>varApply = varApply + 5;</code> 5/5 або 5:5	1
Застосувати отримання частки ділення (%=)	<code>varApply %= 5;</code>	<code>varApply = varApply + 5;</code> 5%5	0
Застосувати узведення до степеня (**=)	<code>varApply **= 5;</code>	<code>varApply = varApply + 5;</code> 5^5	3125
Застосувати додавання одиниці (varApply++)	<code>varApply++;</code>	<code>varApply = varApply + 1;</code>	6
Застосувати віднімання одиниці (varApply--)	<code>varApply--;</code>	<code>varApply = varApply - 1;</code>	4



## Урок 3.6. Функції.

Приклад, як можна створювати функції на JS:

```
function name(a, b, c, ...) {  
    // Перелік команд  
};  
// Де name - це назва функції  
// Де a, b, c, ... - це змінні, які може приймати функція
```

Тепер зробимо функцію, яка приймає значення та показує результат у консолі:

```
function name(a, b, c) {  
    console.log(a + b * c);  
};  
name(5, 10, 4) // У консолі буде: 45, бо 5 + 10 * 4 = 45
```

Але це показати результат у консолі, бо можна ще зробити так, щоб функція повертала результат:

```
var varFun1;  
function name(a, b, c) {  
    return a + b * c; // Повертає значення функції  
};  
varFun1 = name(6, 30, 2)  
// У змінній varFun1 буде: 66, бо 6 + 30 * 2 = 66  
console.log((45, 2, 5)) // У консолі буде: 55
```

А це HTML-код, у якому є поле вводу та кнопка для відображення діалогового повідомлення, який має текст з поля вводу:

```
<!DOCTYPE html>  
<html><head>  
    <meta charset="UTF-8">  
    <title>JavaScript</title>  
</head>  
<body>  
    <input id="input">  
    <button onclick="a()">Alert</button>  
    <script>  
        function a() {  
            const textFromInput = document.getElementById("input")  
            let text = textFromInput.value;  
            // textFromInput.value бере значення з <input>  
            alert(text);  
        }  
    </script>  
</body>  
</html>
```

## Урок 3.7. Рандом.

### Основи генерації випадкових чисел

У JavaScript для генерації випадкових чисел використовується вбудований об'єкт `Math` з методом `random()`. Метод `Math.random()` повертає випадкове число в діапазоні від 0 (включно) до 1 (не включно).

```
let randomNumber = Math.random();
console.log(randomNumber); /* Випадкове число між 0
(включно) та 1 (не включно) */
```

### Генерація випадкових цілих чисел

Щоб згенерувати випадкове ціле число в певному діапазоні, потрібно використовувати `Math.random()` разом з математичними операціями.

Приклад: Випадкове число від `min` до `max` включно:

```
function getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

let randomInt = getRandomInt(1, 10);
console.log(randomInt); // Випадкове число між 1 та 10 включно
```

- `Math.random()` генерує число від 0 до <1.
- `Math.random() * (max - min + 1)` розширює діапазон до `[0, max - min + 1)`.
- `Math.floor()` округлює число до найближчого меншого цілого.
- Додавання `min` зсуває діапазон до `[min, max]`.

### Генерація випадкових чисел з плаваючою комою

Щоб отримати випадкове число з плаваючою комою в конкретному діапазоні, можна використовувати `Math.random()` без округлення.

Приклад: Випадкове число від `min` до `max`

```
function getRandomFloat(min, max) {
    return Math.random() * (max - min) + min;
}

let randomFloat = getRandomFloat(1.5, 5.5);
console.log(randomFloat); /* Випадкове число між 1.5
(включно) та 5.5 (не включно) */
```



## Генерація випадкових елементів з масиву

Для вибору випадкового елемента з масиву використовують `Math.random()` разом з `Math.floor()`.

Приклад: Випадковий елемент з масиву:

```
function getRandomElement(array) {  
    let index = Math.floor(Math.random() * array.length);  
    return array[index];  
}  
  
let fruits = ['apple', 'banana', 'cherry', 'date'];  
let randomFruit = getRandomElement(fruits);  
console.log(randomFruit); // Випадковий фрукт з масиву
```

## Швидкі поради

Не використовуйте `Math.random()` для криптографічно безпечних випадкових чисел. Для таких випадків використовуйте `crypto.getRandomValues()`.

```
let array = new Uint32Array(1);  
window.crypto.getRandomValues(array);  
let cryptographicallySecureRandomInt = array[0];
```

Уникайте використання `Math.random()` для важливих функцій безпеки, таких як генерація паролів чи токенів. Краще використовувати спеціалізовані бібліотеки.

## Підсумок

В JavaScript генерація випадкових чисел є досить простою завдяки методу `Math.random()`. Для більш складних випадків, таких як випадковий вибір з масиву або генерація чисел з плаваючою комою, використовуйте комбінування математичних функцій. Для криптографічних потреб використовуйте API `crypto`.

## Урок 3.8. Скрипти для веб-програмування та фреймворки.

Існує кілька популярних фреймворків для JavaScript, які широко використовуються для створення веб-додатків. Ось деякі з них:

### React

**Розробник:** Facebook

React - це бібліотека JavaScript для побудови користувацьких інтерфейсів. Вона дозволяє створювати компоненти, що можуть повторно використовуватися, і керувати станом додатку.

- **Особливості:**

- Використовує JSX для опису UI компонентів.
- Одностороння прив'язка даних.
- Віртуальний DOM для покращення продуктивності.
- Підтримка розширень, таких як React Router для маршрутизації.

Його приклад:

```
import React from 'react';
import ReactDOM from 'react-dom';

function App() {
  return (
    <div>
      <h1>Hello, React!</h1>
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

## 2. Angular

**Розробник:** Google

Angular - це платформа для побудови веб-додатків, яка включає в себе фреймворк, бібліотеки та інструменти.

- **Особливості:**

- Використовує TypeScript як основну мову.
- Двостороння прив'язка даних.
- Модульна архітектура.
- Вбудовані сервіси для HTTP запитів, маршрутизації та інших завдань.

Його приклад:

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `<h1>Hello, Angular!</h1>`,
  styles: []
})
export class AppComponent { }
```

### 3. Vue.js

**Розробник:** Evan You

Vue.js - це прогресивний фреймворк для побудови користувацьких інтерфейсів, який легко інтегрується в існуючі проекти.

- **Особливості:**

- Легка інтеграція з іншими бібліотеками.
- Одностороння та двостороння прив'язка даних.
- Віртуальний DOM.
- Підтримка компонентів і директив.

Його приклад:

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Vue App</title>
</head>
<body>
  <div id="app"></div>
  <script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
  <script src="app.js"></script>
</body>
</html>

// app.js
new Vue({
  el: '#app',
  data: {
    message: 'Hello, Vue!'
  },
  template: '<h1>{{ message }}</h1>'
});
```

## 4. Svelte

**Розробник:** Rich Harris

Svelte - це інструмент для побудови швидких веб-додатків, який відрізняється тим, що виконує більшу частину роботи під час компіляції, а не в браузері.

- **Особливості:**
  - Без віртуального DOM.
  - Простіший синтаксис.
  - Висока продуктивність.
  - Низький розмір пакетів.

Його приклад:

```
<!-- App.svelte -->
<script>
  let name = 'Svelte';
</script>

<main>
  <h1>Hello {name}!</h1>
</main>

<style>
  h1 {
    color: purple;
  }
</style>

<!-- main.js -->
import App from './App.svelte';

const app = new App({
  target: document.body,
});

export default app;
```

## 5. Ember.js

**Розробник:** Ember Core Team

Ember.js - це фреймворк для побудови амбітних веб-додатків з чіткою структурою і конвенціями.

- **Особливості:**

- Потужна система маршрутизації.
- Двостороння прив'язка даних.
- Велика кількість вбудованих інструментів та допоміжних бібліотек.
- Строга організація коду.

Його приклад:

```
<!-- templates/application.hbs -->
<h1>Hello, Ember!</h1>

// app/app.js
import Application from '@ember/application';
import Resolver from 'ember-resolver';
import loadInitializers from 'ember-load-initializers';
import config from 'my-app/config/environment';

export default class App extends Application {
  modulePrefix = config.modulePrefix;
  podModulePrefix = config.podModulePrefix;
  Resolver = Resolver;
}

loadInitializers(App, config.modulePrefix);
```

Кожен з цих фреймворків має свої переваги та особливості, і вибір залежить від конкретних потреб вашого проекту.

Є також код з виставленням значення повзунка. Приклад коду з виставленням значення повзунка є на наступній сторінці.

```
<!-- Виставляє значення повзунка -->
<!DOCTYPE html>
<html><head>
  <meta charset="UTF-8">
  <title>JavaScript</title>
</head>
<body>
  <input type="range" id="input">
  <script>
    const textFromInput = document.getElementById("input")
    textFromInput.value = 70; // Виставляє значення
                              // повзунка на 70%
  </script>
</body>
</html>
```

## IV РОЗДІЛ. ВИКОРИСТАННЯ БАЗИ ДАНИХ.

### Урок 4.1. Що таке база даних.

База даних (БД) — це організований набір даних, які зберігаються і управляються електронними засобами. Вони дозволяють зберігати, шукати, змінювати і витягувати інформацію. Основні компоненти бази даних включають:

1. **Дані:** Інформація, яка зберігається в базі даних.
2. **Система управління базами даних (СУБД):** Програмне забезпечення, яке використовується для управління базою даних, наприклад, MySQL, PostgreSQL, Oracle, SQL Server.
3. **Таблиці:** Основні структури в реляційних базах даних, де дані зберігаються в рядках і стовпцях.
4. **Запити:** Інструменти для витягування даних з бази даних, часто використовуються мови запитів, такі як SQL.
5. **Схема:** Структура, що визначає організацію даних у базі, включаючи таблиці, поля та взаємозв'язки між ними.

Бази даних використовуються для зберігання різноманітної інформації, від особистих даних до комерційних транзакцій і наукових досліджень. Вони є основою для більшості інформаційних систем і додатків, які потребують обробки великих обсягів даних.

### Урок 4.2. Реляційні та нереляційні бази даних.

Реляційні бази даних – база даних, яка має табличний вигляд, а нереляційні бази даних – база даних, яка має текстовий вигляд.

Реляційні бази даних		Нереляційні бази даних								
<table><tr><th>First Name</th><th>Last Name</th></tr><tr><td>Joe</td><td>Biden</td></tr><tr><td>Emmanuel</td><td>Macron</td></tr><tr><td>Justin</td><td>Trudeau</td></tr></table>		First Name	Last Name	Joe	Biden	Emmanuel	Macron	Justin	Trudeau	<pre>{   "First Name":{     "1":"Joe",     "2":"Emmanuel",     "3":"Justin"   },   "Last Name":{     "1":"Biden",     "2":"Macron",     "3":"Trudeau"   } }</pre>
First Name	Last Name									
Joe	Biden									
Emmanuel	Macron									
Justin	Trudeau									



### Урок 4.3. Використання JSON за допомогою JavaScript.

Нехай у нас JSON таблиця буде знаходитись у сторінці, яка має JS-скрипт. Ось код сторінки:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1.0">
  <title>Load JSON Data</title>
</head>
<body>
  <div id="output"></div>

  <script>
    // Вбудовані JSON дані
    const data = {
      "users": [
        { "name": "John Doe", "age": 30 },
        { "name": "Jane Doe", "age": 25 }
      ]
    };

    // Функція для відображення даних на сторінці
    function displayData(data) {
      const outputDiv = document.getElementById('output');
      data.users.forEach(user => {
        const userDiv = document.createElement('div');
        userDiv.textContent = `Name: ${user.name}, Age: ${user.age}`;
        outputDiv.appendChild(userDiv);
      });
    }

    // Відобразити дані після завантаження сторінки
    window.onload = () => displayData(data);
  </script></body></html>
```

## Урок 4.4. Що таке SQL.

SQL (Structured Query Language) — це мова програмування, яка використовується для керування та маніпулювання реляційними базами даних. SQL дозволяє виконувати такі дії:

1. **Запити даних:** Отримання даних з однієї або кількох таблиць бази даних.

```
SELECT * FROM table_name;
```

2. **Вставка даних:** Додавання нових записів у таблиці.

```
INSERT INTO table_name (column1, column2) VALUES (value1, value2);
```

3. **Оновлення даних:** Зміна існуючих записів у таблиці.

```
UPDATE table_name SET column1 = value1 WHERE condition;
```

4. **Видалення даних:** Видалення записів з таблиці.

```
DELETE FROM table_name WHERE condition;
```

5. **Створення і модифікація структури бази даних:** Створення нових таблиць, змінення структури існуючих таблиць тощо.

```
CREATE TABLE table_name (column1 datatype, column2 datatype);  
ALTER TABLE table_name ADD column_name datatype;
```

6. **Керування доступом до даних:** Встановлення прав доступу для користувачів.

```
GRANT SELECT, INSERT ON table_name TO user_name;
```

SQL є стандартом для роботи з реляційними базами даних, такими як MySQL, PostgreSQL, SQLite, Microsoft SQL Server та інші.

## Урок 4.5. Отримання даних, сортування, умови та оператори.

Ми будемо працювати з базою даних country, що містить таблицю country. Давайте подивимось на цю таблицю:

id	name	continent	region	surfacearea	capital	population
1	Japan	Asia	Eastern Asia	377829	Tokyo	126714000
2	Latvia	Europe	NULL	64589	Riga	2424200
3	Mexico	North America	Central America	1958201	Mexico City	98881000
4	Netherlands	Europe	Western Europe	41526	Amsterdam	15864000
5	Portugal	Europe	Southern Europe	91982	Lisbon	9997600
6	Sweden	Europe	Nordic Countries	449964	Stockholm	NULL
7	Ukraine	Europe	Eastern Europe	603700	Kyiv	50456000
8	United States	North America	NULL	9363520	Washington	278357000
9	Thailand	Asia	NULL	513115	Bangkok	61399000
10	Paraguay	South America	North America	406752	Asunción	5496000
11	Philippines	Asia	Southeast Asia	300000	Manila	75967000
12	New Zealand	Oceania	Australia and New Zealand	270534	Wellington	18886000
13	Norway	Europe	Nordic Countries	323877	Oslo	4478500
14	Monaco	Europe	Western Europe	1.50	Monaco	NULL
15	Malta	Europe	Southern Europe	316	Valletta	380200

SELECT - це оператор, який повертає набір даних (вибірку) з бази даних. Для того, щоб використовувати оператор SELECT, ми повинні вказати, що саме ми хочемо вибрати і звідки ми хочемо вибрати.

```
SELECT continent FROM country;
```

У попередньому запиті ми використовували вираз SELECT, щоб отримати *єдиний стовпець continent* з таблиці *country*. Нам потрібно вказати ім'я

стовпця одразу після ключового слова **SELECT**, а після ключового слова **FROM** вказуємо ім'я таблиці, з якої ми хочемо отримати дані.

Давайте дізнаємося трохи більше про цю базу даних. База даних **country** містить одну таблицю з назвою **country**. Давайте подивимось на цю таблицю.

Ця таблиця має **15 рядків**. Іншими словами, ми маємо 15 різних записів для різних країн.

А як щодо стовпців? Тут ми маємо **7 стовпців**, таких як **id**, **name**, **continent**, **region**, **SurfaceArea**, **capital** та **population**.

1. **id** - номер запису у цій таблиці;
2. **name** - назва країни;
3. **continent** - назва континенту, на якому знаходиться країна;
4. **region** - назва регіону країни;
5. **SurfaceArea** - площа поверхні країни;
6. **capital** - столиця країни;
7. **population** - населення країни.

Ми також можемо отримати декілька стовпців за допомогою оператора **SELECT**. Єдина відмінність полягає в тому, що після слова **SELECT** нам потрібно буде вказати кілька імен стовпців, які повинні бути **розділені комою**. Розглянемо приклад, де ми отримуємо **три** стовпці з таблиці країн:

```
SELECT id, name, capital FROM country;
```

id	name	capital
1	Japan	Tokyo
2	Latvia	Riga
3	Mexico	Mexico City
4	Netherlands	Amsterdam
...	...	...
15	Malta	

Зверніть увагу, що запит **SELECT** повертає всі рядки в заданому стовпчику. Проте, що робити, якщо нам не потрібні всі значення зі стовпця (оскільки вони можуть дублюватися), а потрібні лише унікальні значення? Для таких випадків зручно використовувати ключове слово **DISTINCT**. Причому, це ключове слово використовується безпосередньо перед назвами стовпців. Розглянемо приклад:

```
SELECT DISTINCT region FROM country;
```

Результат є на наступній сторінці.

region
null
Australia and New Zealand
Southeast Asia
Central America
Western Europe
Southern Europe
North America
Eastern Europe
Eastern Asia
Nordic Countries

Ми знаємо, що оператор SELECT вибирає всі рядки з таблиці всіх або певних стовпців. Однак, що робити, якщо нам потрібно отримати лише певну кількість рядків, наприклад?

Зауважте. У різних системах керування це реалізовано з різним синтаксисом. Оскільки ми використовуємо PostgreSQL, ми повинні використовувати оператор LIMIT.

У наведеному нижче прикладі ми витягуємо перші 7 рядків (у нашому випадку - столиці країн) зі стовпчика:

```
SELECT capital FROM country LIMIT 7;
```

capital
Tokyo
Riga
Mexico City
Amsterdam
Lisbon
Stockholm
Kyiv

Команда для отримання усіх даних з таблиці:

```
SELECT * FROM country;
```

Давайте зрозуміємо, що означає "групування даних", на простому прикладі таблиці працівників:

department	employee_id	first_name	hire_date	last_name	salary
Engineering	1	John	2015-03-01T00:00:00Z	Doe	80000.00
Engineering	2	Jane	2017-08-15T00:00:00Z	Smith	90000.00
Marketing	3	Alice	2016-11-10T00:00:00Z	Johnson	75000.00
Marketing	4	Bob	2018-06-25T00:00:00Z	Brown	72000.00
...	...	...	...	...	...
Sales	10	James	2017-05-18T00:00:00Z	Clark	

Тепер давайте уявимо, що у нас є завдання "визначити кількість працівників в кожному департаменті." Для цього ми згрупуємо дані за стовпцем department і використаємо агрегацію за допомогою функції COUNT(\*).

```
SELECT department, COUNT(*) AS number_of_employees
FROM employees
GROUP BY department;
```

Отже, як ви можете побачити, синтаксис для групування даних виглядає так:

```
SELECT column1, AGG_FUNC(column2)
FROM table
GROUP BY column1;
```

Примітка: AGG\_FUNC означає агрегатні функції на кшталт MAX, MIN, COUNT тощо.

Цей синтаксис існує для пошуку певних значень за допомогою агрегатних функцій у специфічних колонках.

Розгляньмо інший приклад: наше завдання полягає в знаходженні відділу з найвищою середньою зарплатою.

Щоб отримати такі дані, нам потрібно групувати дані за колонкою department та потім використати функцію AVG(), щоб розрахувати середню зарплату:

```
SELECT department, AVG(salary) as average_salary
FROM employees
GROUP BY department;
```

Примітка: Зверніть увагу, що ми не будемо використовувати цю таблицю в завданнях; таблиця `employees` буде використовуватися виключно для демонстрації прикладів синтаксису та їх застосування.

У цьому курсі ми будемо працювати з базою даних системи монреальського метро, яка містить таблицю `metro_travel_time`.

Ця таблиця міститиме інформацію про лінію станції(`line_name`), її назву(`station_name`), та кількість часу, який потрібен поїзду для подорожі від однієї станції до наступної(`time_to_next_station`).

Ось як виглядає ця таблиця та попередній перегляд даних у ній:

id	line_name	station_name	time_to_next_station
1	Green	Angrignon	10
2	Green	Monk	16
3	Green	Verdun	9
4	Green	Charlevoix	17
...	...	...	...
21	Yellow	Longueuil	10

Як ви можете бачити, це не складна таблиця. Давайте подумаємо, де ми можемо застосувати групування тут.

Найочевидніший варіант - це групування за кольорами ліній метро. Це означає, що ми можемо агрегувати дані, групуючи їх за кольором лінії метро.

Тепер давайте попрактикуємось у групуванні, виконуючи завдання.

Для статистичного аналізу нам було доручено порахувати кількість станцій на кожній лінії та впорядкувати їх за зростанням кількості станцій на кожній з ліній метро.

Для цього нам потрібно знайти кількість станцій на кожній з ліній метро, а потім відсортувати їх від найменшої кількості станцій до найбільшої.

Таким чином, будівельна компанія зрозуміє, яким лініям метро слід надавати першочергову увагу для додавання станцій.

Для нас важливо зрозуміти порядок запису кюз, зокрема, де має бути розміщений GROUP BY кюз.

Отже, порядок виглядає так:

1. Оператор SELECT;
2. FROM table;
3. Конструкція WHERE;
4. Конструкція GROUP BY;
5. Конструкція ORDER BY;
6. Конструкція LIMIT.

З цього порядку очевидно, що оператор GROUP BY повинен бути написаний ПІСЛЯ оператора WHERE (або після FROM таблиці, якщо у вашому запиті не використовується фільтрація з використанням SELECT) та також ДО оператора ORDER BY.

Давайте розглянемо приклад такого порядку інструкцій, використовуючи нашу таблицю employee. Припустимо, нам потрібно отримати кількість працівників у кожному department, чия salary є вищою за 70000, та відсортувати їх за зростанням:

```
SELECT department, COUNT(employee_id) AS number_of_employees
FROM employees
WHERE salary > 70000
GROUP BY department
ORDER BY number_of_employees;
```

Примітка: Варто зазначити, що клас LIMIT завжди пишеться в кінці. Так ви легко запам'ятаєте його розташування у запиті.

Будівельна компанія вирішила збільшити кількість станцій на Жовтій лінії метро.

Наше наступне завдання - знайти час обороту для кожної лінії. Для компанії важливо мати можливість закрити Жовту лінію для технічного обслуговування та розширення шляхом додавання нових станцій метро, тому вкрай важливо не створювати надмірні незручності для пасажирів.

Тому нам потрібно визначити загальний час обороту потяга, підсумувавши час до кожної станції (використовуючи функцію SUM()).

Примітка: Якщо ми просто підрахуємо суму часу до кожної станції, це буде час руху потяга від однієї кінцевої станції до іншої. Однак також



важливо знати загальний час обороту потяга по всій лінії метро. Щоб досягти цього, ми повинні помножити суму на 2.

Щоб зрозуміти, як виконати це завдання, розгляньмо приклад з таблицею employees.

Припустимо, нам потрібно знайти відділ з найвищою середньомісячною зарплатою.

Для цього можемо використати такий запит:

```
SELECT department, AVG(salary) / 12 AS average_monthly_salary
FROM employees
GROUP BY department
ORDER BY average_monthly_salary DESC;
```

Отже, ми **отримуємо необхідні дані** в результаті.

Ось випадок: школа рекомендували вас, і школа зацікавилася вами, оскільки в них також є кілька завдань для вас. Але спершу давайте ознайомимось з таблицею, яку вони надали:

id	student_surname	class_letter	subject_name	grade
1	Smith	A	Mathematics	85
2	Johnson	B	Physics	90
3	Williams	C	Chemistry	78
4	Brown	A	Biology	92
...	...	...	...	...
100	Gonzales	A	Biology	

Як ви можете побачити, у школі навчається загалом **100 учнів**, інформація про яких надана у даній таблиці. В стовпчику class\_letter є інформація, яка може мати 3 варіанти: A, B або C. Також тут вказано назву предмету (subject\_name) та оцінку учня (grade). Таблиця проста і містить оцінки за іспити з різних предметів.

Подивимось, скільки учнів знаходиться в кожному класі за допомогою наступного запиту:

```
SELECT class_letter, COUNT(DISTINCT student_surname) AS anumber_of_students
FROM student_grades
GROUP BY class_letter;
```

Школа задоволена нашою роботою і погоджується продовжити співпрацю.

Тепер у них є для нас нове завдання. Топ-10 учнів з найвищим середнім балом отримують в якості винагороди поїздку до наукового центру. Однією з обов'язкових умов є отримання оцінки вище 90 з математики на іспиті. Щоб знайти таких студентів, вони звернулися до вас.

Подивимось, що нам потрібно зробити, використовуючи нашу таблицю employee як приклад.

Припустимо, нам потрібно дізнатися, у яких департаментах є працівники, які були прийняті на роботу до 2019 року та середня заробітна плата в цих департаментах. Для виконання такого завдання можна використати наступний запит:

```
SELECT department, AVG(salary) AS average_salary
FROM employees
WHERE hire_date < '2019-01-01'
GROUP BY department;
```

Як ви можете побачити, таких працівників всього 3, і ми використали необхідні інструменти для досягнення цього результату. Ваше завдання буде дуже схожим, я впевнений, що ви з цим впораєтесь!

Ось попередній перегляд таблиці student\_grades, з якою ми працюємо:

id	student_surname	class_letter	subject_name	grade
1	Smith	A	Mathematics	85
2	Johnson	B	Physics	90
3	Williams	C	Chemistry	78
4	Brown	A	Biology	92
...	...	...	...	...
100	Gonzales	A	Biology	

Школа дуже вдячна за вашу роботу, і тепер у нас є нове завдання.

Виявилось, що деякі студенти **складали додаткові іспити**, коли мали скласти лише **один**. Школа підозрює їх у **нечесності**, оскільки кожен студент має мати лише одну оцінку.

Нам доручили **з'ясувати прізвища цих студентів** і передати їх адміністрації школи, щоб вони змогли вжити необхідних заходів.

Давайте разом подумаємо **як ми можемо це зробити**. Ви можете почати з того, що ми можемо зробити це за допомогою клавзи WHERE, і це буде виглядати приблизно так:

```
SELECT student_surname
FROM student_grades
WHERE COUNT(grade) > 1;
```

Але, як ви можете побачити, ми отримуємо помилку, що вказує на те, що ми не можемо використовувати агрегатні функції усередині клози WHERE. Ось де нам знадобиться клоза HAVING.

Давайте зрозуміємо, що це таке і як це використовувати, на прикладі з нашої таблиці employee.

Припустимо, нам потрібно отримати відділи, де середня заробітна плата співробітників нижче \$70,000 на рік.

Для цього нам знадобиться використовувати агрегатну функцію та клозу HAVING.

Давайте подивимося, як ми можемо це зробити:

```
SELECT department
FROM employees
GROUP BY department
HAVING AVG(salary) < 70000;
```

Ми отримали один відділ у відповідь, використовуючи клас HAVING, де ми встановили умову для колонки, по якій групували дані.

Примітка: Щоб використовувати агрегацію даних в класі HAVING, нам потрібно мати групування даних у нашому запиті. Як у запиті вище, ми групували дані по колонці department.

Давайте розглянемо більш узагальнений синтаксис класу HAVING та коли його найкраще використовувати:

```
SELECT column1, column2... --(за бажанням)
FROM table
GROUP BY column1
HAVING AGG(column_n) --(умова);
```

Давайте також коротко розглянемо головну відмінність між клозами WHERE та HAVING та коли варто використовувати кожен з них:

Клоз WHERE використовується до агрегації даних, тоді як клоз HAVING застосовується після агрегації даних;

Клоз WHERE пишеться до GROUP BY, у той час як клоз HAVING записується після GROUP BY.

Раніше школа проводила **змагання** для учнів, які складали **Математику**, і деякі з них отримали нагороди. Тепер школа хоче переконатися, що не буде учнів, які **обманюють** та складають **більше одного** іспиту, включаючи іспит з математики.

Тому школа попросила нас **визначити прізвища** тих учнів, які склали **більше одного** іспиту, одним з яких був Mathematics .

Ось запит з нашого **попереднього завдання**, який ви можете використати як приклад:

```
SELECT student_surname, AVG(grade) as average_grade
FROM student_grades
GROUP BY student_surname
HAVING COUNT(grade) > 1;
```

Де ми вказували знак «>» - це знак нерівності. Ось як можна задавати знаки нерівності у SQL при умові:

Знак	Що означає
=	Дорівнює
<>	Не дорівнює
>	Більше
<	Менше
>=	Більше або дорівнює
<=	Менше або дорівнює

Раніше ми працювали на різних компаніях та виконували запити SELECT для їхніх потреб. Проте нам потрібно навчитися створювати та модифікувати таблиці.

Давайте перейдемо безпосередньо до справи!

Таблиці створюються за допомогою оператора CREATE, який має схожу структуру до оператора SELECT, але замість вибірки даних, він створює дані.

Подивимося на синтаксис:

```
CREATE TABLE example (  
    id INT PRIMARY KEY,  
    some_info VARCHAR(50)  
);
```

Примітка: коли ви запускаєте ці приклади, ви не отримаєте жодного виводу, оскільки ці приклади лише створюють нову таблицю. Якщо ви спробуєте запустити код знову, ви отримаєте повідомлення про помилку, яке говоритиме, що таблиця вже існує. Ці фрагменти коду є прикладами, і пізніше, в завданні, у ці новостворені таблиці будуть вставлені дані та відображені на екрані, щоб ви могли побачити, що все працює.

Тепер розберемось, що написано вище.

Цей запит створить ПОРОЖНЮ таблицю із двома колонками: id та some\_info.

Зверніть увагу на типи даних, які використовуються. Слова INT чи VARCHAR позначають тип даного для кожної колонки. Наприклад, INT представляє цілочисельні дані, тоді як VARCHAR(50) представляє текст з максимумом 50 символів.

Зараз ми не будемо заглиблюватися у всі типи даних, оскільки їх доволі багато. Ми зосередимося на основних типах даних у цьому розділі, і ми розглянемо кожен з них по мірі того, як будемо просуватися в навчанні!

Наприклад, створимо іншу таблицю з різними типами даних:

```
CREATE TABLE users (  
    id INT PRIMARY KEY,  
    name VARCHAR(50),  
    birthdate DATE,  
    salary DECIMAL(10, 2),  
    is_active BOOLEAN  
);
```

За допомогою цього запиту ми створюємо порожню таблицю, яка має містити інформацію про користувачів, включаючи:

1. ID з типом даних ціле число;
2. Інформацію про ім'я, з типом даних VARCHAR(50);
3. Інформацію про дату народження, з типом даних DATE.
4. Інформацію про зарплату, з типом даних число з плаваючою комою;
5. Чи є користувач активним, з типом даних, що приймає лише значення true або false.

Обмеження:

Ви могли помітити, що поруч з кожним значенням ID ми поміщаємо слова PRIMARY KEY. Це називається обмеженням і означає обмеження, накладене на цю колонку.

Наприклад, PRIMARY KEY забезпечує унікальність та ідентифікує кожен рядок у таблиці. В таблиці може бути тільки одна така колонка.

Також існують інші обмеження, наприклад:

- NOT NULL: Гарантує, що колонка не буде містити значень NULL;
- UNIQUE: Забезпечує унікальність усіх значень у колонці чи комбінації колонок;
- DEFAULT: Встановлює значення за замовчуванням для колонки, якщо при вставці даних для цієї колонки не вказано значення.

Це не всі обмеження, які використовуються, але поки що нам потрібно саме ці.

Розглянемо приклад, де ми модифікуємо попередню таблицю:

```
CREATE TABLE users_2 (  
    id INT PRIMARY KEY,  
    name VARCHAR(50) NOT NULL,  
    birthdate DATE,  
    salary DECIMAL(10, 2) DEFAULT 50000,  
    is_active BOOLEAN  
);
```

Тепер стовпець name не може мати порожніх або null значень, а стовпець salary за замовчуванням має значення 50000.

Таким чином, ви можете використовувати обмеження для контролю стовпців таблиці під час її створення.

У попередньому розділі ми навчились створювати таблиці.

Але уявімо ситуацію, коли нам потрібно додати стовпець до існуючої таблиці. Було б досить безглуздо видаляти таблицю (особливо якщо вона вже містить якісь дані) і потім створювати нову, знову заповнюючи її даними.

Тому в цьому розділі ми розглянемо операцію ALTER.

Давайте подивимося, як використовувати цю операцію:

```
CREATE TABLE library (  
  id INT PRIMARY KEY,  
  title VARCHAR(50) NOT NULL,  
  author VARCHAR(50),  
  pages INT  
);  
  
ALTER TABLE library ADD price DECIMAL DEFAULT 300;  
  
ALTER TABLE library DROP COLUMN price;
```

Як ви можете побачити, це скрипт для створення таблиці з попереднього розділу.

Далі йдуть дві операції ALTER. Перша операція додає до таблиці стовпець price, встановлюючи значення за замовчуванням 300 для цього стовпця. Друга операція видаляє цей стовпець.

Синтаксис надзвичайно простий:

```
ALTER TABLE table_name ADD/REMOVE column_name DATA_TYPE;
```

Синтаксис насправді досить простий.

Примітка! За допомогою оператора ALTER ви можете виконувати різні операції на рівні схеми в таблиці, наприклад, додавання або видалення обмежень, перейменування, зміну типів даних та додавання чи видалення індексів.

Перейдемо до іншої операції, а саме операції вставки. Для вставки даних в SQL можна використовувати оператор INSERT. Для використання INSERT ми маємо вказати, в які колонки ми хочемо додати значення.

Ось як виглядає синтаксис цього оператора:

```
INSERT INTO library (id, title, author, pages) VALUES
  (1, 'CAMINO GHOSTS', 'John Grisham', '213'),
  (2, 'FUNNY STORY', 'Emily Henry', '341');
```

Ви мабуть правильно помітили, що це **уришок скрипту з попереднього розділу**, де **дані вставляються** в таблицю library.

Давайте розберемо, що тут відбувається:

1. Спочатку пишуться ключові слова INSERT INTO, за якими слідує table\_name, де будуть вставлені дані;
2. Потім **відкриваються дужки**, і вказуються **назви стовпців**, куди будуть вставлені дані; у нашому випадку є 4 стовпці;
3. Після цього пишеться ключове слово VALUES, і **відкриваються дужки**, куди будуть записані дані;
4. Дані слід записувати в **тому ж порядку, що й назви стовпців**, при цьому потрібно дотримуватися типів даних. Наприклад, ви не можете вставити цілочисельне значення в стовпець з типом даних VARCHAR;
5. **Дужки закриваються**, а потім ставиться кома, таким чином заповнюється один рядок. Ви можете заповнити **стільки рядків, скільки вважаєте за потрібне**, використовуючи цей метод.

Підсумовуючи, загальний синтаксис оператора INSERT виглядає так:

```
INSERT INTO table_name (column1_name, column2_name) VALUES
  (column1_value, column2_value),
  (column1_value, column2_value),
  ...;
```

Не забудьте про крапку з комою в кінці!

Варто згадати ще дві операції в DDL: DROP та TRUNCATE.

Давайте коротко розглянемо, що робить кожна з цих операцій:

**DROP:** Використовується для видалення об'єктів бази даних, таких як таблиці, бази даних та індекси.

**TRUNCATE:** Видаляє всі рядки з таблиці, але зберігає її структуру.



Я використовував ці операції для очищення або видалення таблиць, щоб перевірити завдання в попередніх розділах. Їхня синтаксис досить простий; давайте на них подивимось:

```
ALTER TABLE table_name ADD/REMOVE column_name DATA_TYPE;
```

Цей код видалить таблицю employees з бази даних. У багатьох СУБД ця операція вимагає певних дозволів, і якщо ви працюєте над проектом, у вас може не бути доступу до такої операції. Ви дізнаєтеся про ролі та як їх налаштувати в наступному курсі, який охоплює Розширені концепції SQL.

```
ALTER TABLE table_name ADD/REMOVE column_name DATA_TYPE;
```

Цей код видалить усі рядки із таблиці employees, повністю очистивши її та зробивши порожньою. Ця операція не впливатиме на структуру таблиці, тобто не змінить колонок або обмежень. Вам також потрібні дозволи в СУБД для такої операції, оскільки не кожному мають бути надані здібності очищати таблицю.

Використовуйте ці операції обережно, тому що якщо у вас немає знімків бази даних, ви не зможете відкотити видалення таблиці або обрізку рядків.

Примітка! Часто розробники використовують м'яке видалення, додаючи нову колонку is\_deleted з типом даних BOOLEAN, і коли деякі рядки видаляються, статус встановлюється як true (або 1). Таким чином, ви можете бачити видалені дані та не хвилюватися про їх цілісність.

Ви вже знаєте, як очистити таблицю, додати стовпець, вставити дані тощо. Однак для правильної взаємодії з базою даних нам потрібно розуміти, як оновлювати та видаляти окремі рядки.

Для цього існують два твердження та типи запитів: запити UPDATE та DELETE.

UPDATE: Використовується для модифікації існуючих даних у таблиці. За допомогою такого запиту ми можемо змінити дані в таблиці, не впливаючи на інші рядки. Давайте розглянемо приклад з таблицею medications, яка є на наступній сторінці:

id	name	price
1	Aspirin	5.00
2	Ibuprofen	10.00
3	Paracetamol	8.00
4	Naproxen	12.00
...	...	...
10	Azithromycin	22.00

Уявімо, що нам потрібно оновити ціну на певний тип лікарства. Наприклад, зараз на Ibuprofen діє 50% знижка, і ми повинні змінити ціну на цей продукт.

Наш запит на оновлення буде виглядати так:

```
UPDATE medications
SET price = 4
WHERE id = 2;

SELECT *
FROM medications
ORDER BY id;
```

Ось, ми оновили таблицю medications, щоб ціна продукту з id 2 (Ibuprofen) була встановлена на 4. Після цього ми вибрали всі стовпці з таблиці, щоб переконатися, що стовпець ціна успішно оновлено. Ви можете замінити будь-яке значення і побачити, як працює операція оновлення в SQL.

У підсумку, загальний синтаксис виглядає так:

```
UPDATE table_name
SET column_name = значення
WHERE якась_умова;
```

Операція команди DELETE майже ідентична за принципом. Однак тут ми не використовуємо ключове слово SET, адже ми нічого не змінюємо; ми просто видаляємо рядки.

Синтаксис для видалення виглядатиме так:

```
DELETE FROM table_name
WHERE some_condition;
```

Але я нагадаю вам, що видалення рядків слід проводити обережно, адже ви не зможете легко їх відновити.

Примітка! Якщо ви не включите умову WHERE, дані будуть оновлені або видалені для усіх рядків.

