

# Веб- разработка

## I РАЗДЕЛ. СОЗДАНИЕ СОБСТВЕННЫХ ВЕБ-СТРАНИЦ. (стр. 4 – 20)

- 1.1. Из чего состоит веб-страница. (стр. 4)
- 1.2. Что такое HTML. (стр. 4)
- 1.3. Невизуальные компоненты. (стр. 4 – 6)
- 1.4. Написание текста. (стр. 6 – 8)
- 1.5. Атрибуты. (стр. 9 – 10)
- 1.6. Добавление кнопок, флажков, переключателей и т.д. (стр. 10 – 12)
- 1.7. Добавление изображений. (стр. 13)
- 1.8. Создание таблиц. (стр. 13)
- 1.9. Создание нумерованных и маркированных списков. (стр. 14)
- 1.10. Создание выпадающих списков. (стр. 15)
- 1.11. Добавление видео и аудио контента. (стр. 15 – 16)
- 1.12. Использование блоков: <div>, <footer>, <header>, и других. (стр. 16 – 19)
- 1.13. Выкладка сайта в сеть. (стр. 19 – 20)

## II РАЗДЕЛ. СТИЛИЗАЦИЯ СТРАНИЦ. (стр. 21 – 50)

- 2.1. Что такое CSS. (стр. 21)
- 2.2. Использование CSS в HTML. (стр. 21 – 24)
- 2.3. Стилизация классов и ID. (стр. 24)
- 2.4. Общие настройки. (стр. 25 – 30)
- 2.5. Display-опция. (стр. 31 – 33)
- 2.6. Фильтры. (стр. 33 – 36)
- 2.7. Flexbox. (стр. 36 – 45)
- 2.8. Анимация. (стр. 45 – 48)
- 2.9. Адаптация. (стр. 48 – 50)

## III РАЗДЕЛ. Скрипты. (стр. 50 – 72)

- 3.1. Что такое JavaScript. (стр. 51)
- 3.2. Добавление и исполнение скриптов на веб-странице. (стр. 51 – 52)
- 3.3. События. (стр. 52 – 56)
- 3.4. Первые скрипты. (стр. 56 – 60)
- 3.5. Операторы, условия, разветвления и циклы. (стр. 60 – 64)
- 3.6. Функции. (стр. 64 – 65)
- 3.7. Рандом. (стр. 65 – 66)
- 3.8. Скрипты для веб-программирования и фреймворки. (стр. 67 – 72)

## IV РАЗДЕЛ. ИСПОЛЬЗОВАНИЕ БАЗЫ ДАННЫХ. (стр. 73 – 91)

- 4.1. Что такое база данных. (стр. 73)
- 4.2. Реляционные и нереляционные базы данных. (стр. 73)
- 4.3. Использование JSON с помощью JavaScript. (стр. 74)
- 4.4. Что такое SQL. (стр. 75)
- 4.5. Получение данных, сортировка, условия и операторы. (стр. 76 – 91)

Данное руководство связано с веб-программированием. В этой книге вы научитесь: создание веб-страниц; стилизация страниц; программирование на языке JavaScript; использовать базу данных с помощью языка программирования SQL. Эти языки очень популярны в веб-разработке и первым начинают изучать веб-разработчики – использование HTML, который есть на всех веб-страницах.

## I РАЗДЕЛ. СОЗДАНИЕ СОБСТВЕННЫХ ВЕБ-СТРАНИЦ.

### Урок 1.1. Из чего состоит веб-страница.

В общей сложности страницы состоят из элементов, стилизации и функциональности. За объекты отвечает язык гипертекстовой разметки HTML, за стилизацию отвечает язык каскада стилей CSS, а за функциональностью отвечают скрипты языков программирования.

### Урок 1.2. Что такое HTML |

HTML (Hyper Text Marker Language) – язык гипертекстовой разметки, отвечающий за наличием элементов на сайте. Вот написание кода на HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- <head> - тег, отвечающий за наличие
    элементов функций Где lang="en" - указание
    на язык страницы, чтобы браузер понимал, на каком
    языке там написано.
    <!DOCTYPE html> - указание использования
    версии HTML (В данном случае, это: HTML 5.0)
    -->
  </head>
  <body>
    <!-- <body> - тег, отвечающий за наличие
    визуальных элементов -->
  </body>
</html>
```

Где писалось<!-- какой-то текст --> - это комментарий, не влияющий на работу страницы.

Также следует помнить, что: <html>; <body>; <head> – это все теги. Они могут быть с закрывающим тегом и без него. К примеру, теги: <body>; <head>; <html> - открывающие теги, а теги, имеющие в начале своего названия символ "/", такие как: </head>; </body>; </html> - закрывающие теги. О закрывающих и открывающих тегах не следует забывать.

### Урок 1.3. Невизуальные компоненты.

В общем, невидимые компоненты это те, которые мы не видим на веб-странице, но выполняют функцию, такие как: импорт скриптов; изменение общего названия страницы; сохранение информации о странице; тому

подобное. Для использования невизуальных компонентов, нужно их использовать в тезисе <head>. Вот код с использованием этих компонентов:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Установление общей
названия страницы -->
    <title>Название страницы</title>

    <!-- Импорт стилизации -->
    <link rel="stylesheet" href="файлСтиляСтраницы.css">

    <!-- Задание иконки сайта -->
    <link rel="icon" type="image/png" href="./assets/favicon.png">

    <!-- Информация о странице, которую можно найти за
ключевыми словами в атрибуте "content". Где атрибут
"name" в котором значение "keywords" - это указание на
выполнение функции тега <meta>. В данном случае -
было указано о функции поиска по ключевым словам по поисковой системе-->
    <meta name="keywords" content="Википедия, Метатег, статья">

    <!-- Информация об имени автора страницы -->
    <meta name="author" content="Вадим Савенко">

    <!-- Аналог "author" где можно еще указать язык -->
    <meta name="copyright" lang="en" content="Вадим Савенко">

    <!-- Информация о странице (Его описание) -->
    <meta name="description" content="Страница HTML">

    <!-- Информация о программе, которой использовалось для
создание данной веб-страницы
    <meta name="generator" content="Macromedia Dreamweaver 4.0">

    <!-- Информация о языке страницы -->
    <meta http-equiv="content-language" content="en">

    <!-- Информация о -->
    <meta http-equiv="Content-Style-Type" content="text/stylus-lang">
    <!-- можно написать следующим образом -->
    <meta http-equiv="Content-Style-Type" content="text/less">

    <!-- Информация о кодировании страницы -->
    <meta charset="UTF-8">
    <!--Продолжение кода на следующей странице -->
```

```

<!-- Информация о том, что данную страницу нужно открывать
во фрейме -->
<meta http-equiv="Window-target" content="_top">

<!-- ICMB: позволяет привязать всю страницу к географическим
координат -->
<meta name="ICBM" content="12.345, 67.890">

<!-- Отметьте группу geo. выполняет аналогичную задачу, позволяя
явно указать название населенного пункта и регион ->
<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, отвечающий за наличие
визуальных элементов -->
</body>
</html>

```

Это все невидимые компоненты, которые нужно знать каждому, но были показаны все необязательные компоненты. Вот код с обязательными компонентами:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Название страницы</title>
    <link rel="stylesheet" href="файлСтиляСтраницы.css">
    <link rel="icon" type="image/png" href="favicon.png">
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- <body> - тег, отвечающий за наличие
    визуальных элементов -->
  </body>
</html>

```

**Примечание!**Если не использовать <title>, то общее название страницы будет называться файлом кода этой страницы.

## Урок 1.4. Написание текста.

Для страницы, текст — визуальный компонент, который имеет несколько символов (Это: буквы; точка; запятая; точка с запятой; двоеточие; и т.д.). Код с тегами текста находится на следующей странице.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- <head> - тег, отвечающий за наличие
         элементов функций (невизуальных компонентов).-->
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- Обычный текст -->
    <p>
      pppp p.
      pppp p.
    </p>

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

    <!-- Заголовок 1-ого уровня -->
    <h1>Эта страница о тексте.</h1>

    <!-- Заголовок 2-ого уровня -->
    <h2>Эта страница о тексте.</h2>

    <!-- Заголовок 3-го уровня -->
    <h3>Эта страница о тексте.</h3>

    <!-- Заголовок 4-ого уровня -->
    <h4>Эта страница о тексте.</h4>

    <!-- Заголовок 5-ого уровня -->
    <h5>Эта страница о тексте.</h5>

    <!-- Заголовок 6-ого уровня -->
    <h6>Эта страница о тексте.</h6>

    <!-- гиперссылка. Где данные в "href",
         там ссылку на страницу -->
    <a href="https://website/">Эта страница о тексте.</a>

    <!-- Жирный текст -->
    <b>Эта страница о тексте.</b>
    <!-- Продолжение кода на следующей странице -->
```

```

    <br> <!--Переводит элементы и текст на другой срок-->

    <!-- Наклоненный текст -->
    <i>Эта страница о тексте.</i>

    <!-- Подчеркнутый текст -->
    <u>Эта страница о тексте.</u>

    <hr> <!-- Создает полосу -->

    <!-- Зачеркнутый текст -->
    <s>Эта страница о тексте.</s>

    <!-- Подстрочный текст -->
    <sub>Эта страница о тексте.</sub>

    <!-- Сверхсрочный текст -->
    <sup>Эта страница о тексте.</sup>
</body>
</html>

```

В случае проверки кода в браузере, будет разница между 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. Атрибуты.

Атрибут – емкое, необходимое для обеспечения целостности объекта, субъекта (предмета), свойство, его часть, приложение. Атрибуты – это поля данных, относящиеся к файлу, но не являющиеся его частью. Они не учитываются при подсчете размера файла и могут быть скопированы или удалены без изменения файла. Система использует атрибуты для хранения, например, размера, типа файла, даты его последнего изменения и т.д. Так же и в других операционных системах. Отличие заключается в том, что Вы можете добавить любой атрибут к любому файлу, затем отобразить и сделать его редактируемым в окне Tracker. Вам нужно определить какого типа атрибут Вы хотите добавить в файл (например: лента, целое число и время), дать определение и описание. Вопрос для чего нужен атрибут → он указывает путь или маршрут браузера, атрибут указывает тегу куда разместить текст, который должен быть список и т.д. Какие бывают атрибуты → конкретный нужный для конкретного тега и общий или общий, используемый для абзаца, для заголовка, для таблицы. Ну что же в принципе все, есть атрибуты, которые можно добавлять к любому тегу, а есть конкретные, которые добавляют к конкретному тегу, добавлю, что атрибут “align” считают устаревшим, но так можно сказать, что сам язык гипертекстовой разметки HTML устаревший, и не нужна когда присутствуют системы управления контентом.

Атрибуты всегда записываются внутри тега, после них следует знак равенства и детали атрибута, заключенные в двойные кавычки. Точка с запятой после атрибута служит для разделения команд разных стилей. Вот так записывается тег с атрибутами:

```
<тег атрибут Первый="" атрибутВторой=""></тег>
```

Примечание. Тег может иметь несколько атрибутов. Атрибуты не являются обязательными для использования с тегами, но есть некоторые атрибуты, которые следует использовать для функциональности или стилизации тега. Вот все атрибуты, используемые в тегах:

**Атрибуты, являющиеся всеми визуальными компонентами:** class; id; style; width; . Для примера возьмем тэг <p>. Вот написание этого тега с использованием этих атрибутов:

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

*class*– служит для задания класса тега.

*id*– служит для задания ID для тэга.

*style*– служит для задания CSS стиля для тэга.

*width*— служит для задания ширины компонента в процентах от размера страницы в окне обозревателя или в пикселях. Например: `width=10px`; `width="10%"`.

*align*— служит для выставления компонентов горизонтально с центральной (`align="center"`), правой (`align="right"`) или левой стороны (`align="left"`).

**Атрибуты `src` и `href`:** атрибуты, служащие для ссылки на локальный файл (в том числе и страницы) или сетевой файл:

```

```

Разница между атрибутами `src` и `href`:

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

Для ссылки на адрес нужно указывать следующим образом:

- **На веб-страницу:**

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

- **На электронную почту:**

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

- **На номер телефона:**

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

**Атрибуты `alt`:** атрибут, вставляющий текст вместо изображения, если изображение не удалось загрузить. Этот тег используется только с тегом `<img>`. Вот его написание с этим тегом:

```

```

**Атрибуты `target`:** атрибут, используемый только в тезисе `<a>` и открывающий ссылку на новой вкладке, если `target="_blank"`. К примеру:

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

**Атрибуты `download`:** атрибут, используемый только в тезисе `<a>` и дающий команду браузеру, что ссылку на файл можно только загружать. К примеру:

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

**Атрибуты типа и другие:** об этих атрибутах будет разговор в других затем.

## Урок 1.6. Добавление кнопок, флажков, переключателей и т.д.

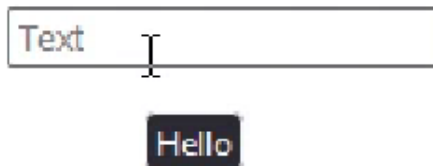
Тег `<button>` отвечает за создание кнопки, отвечающей за выполнение функции при нажатии на нее. Вот написание кода с этим тегом:

```
<button>Кнопка</button>
```

Также есть тег `<input>`, отвечающий за создание поля ввода. Вот написание кода с этим тегом:

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

В этом случае на странице будет:

The image shows a web form element. At the top is a rectangular 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. Below the input field is a small, dark blue button with the word "Hello" written in white text.

Но с этим тегом можно делать не только кнопки, но и другие компоненты с помощью атрибута `type`. Вот примеры использования с атрибутом `type`:

type	Компоненты
"text"	Текстовое поле
"email"	Поле почтового адреса
"tel"	Поле номера телефона
"button"	Кнопка
"checkbox"	Флажок
"number"	Числовое поле
"date"	Поле даты
"time"	Поле времени
"datetime-local"	Поле даты и времени
"password"	Поле пароля
"color"	Кнопка для выбора цвета
"file"	Кнопка для выбора файла
"radio"	Переключатель (Радио-кнопка)
"range"	Ползунок

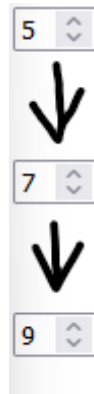
Также тег `<input>` имеет необязательные атрибуты:

- ***min***—минимальное допустимое количество символов или значение в компоненте
- ***max***—максимальное допустимое количество символов или значение в компоненте
- ***step***—на сколько шагов будет перемещаться ползунок или числовое поле.

Использование их в коде страницы:

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

Результат кода в браузере:

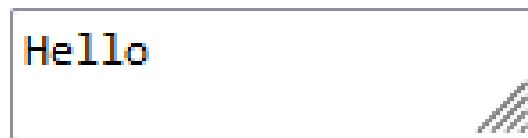


Эти атрибуты могут использоваться с типами: "time"; "range"; "number".

Также существует тег `<textarea>`, который отвечает за создание поля ввода, в котором можно изменять его размер. Вот его написание в коде:

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

При открытии страницы в браузере:



Также этот тег имеет атрибут «rows», который отвечает за высоту этого тега по строкам. Если не указывать этот атрибут, то его высота будет равна двум строкам. Пусть будет случай, когда нужно произвести высоту этого компонента будет на 5 строк. Тогда вот пример при использовании этого атрибута:

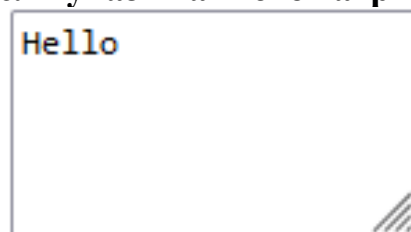
```
<textarea rows="5">Hello</textarea>
```

При открытии страницы в браузере:

Если не указывать этот атрибут



Если указывать этот атрибут



## Урок 1.7. Добавление изображений.

Для добавления изображений на страницу HTML необходимо использовать тег `<img>`, который имеет атрибуты `alt` и `src`. Вот код с этим тегом:

```
  

```

- `alt` служит для вставки текста вместо изображения в случае, когда загрузка изображения была неудачной попыткой.
- `src` служит для задания изображения для тега `<img>`.

## Урок 1.8. Создание таблиц.

Вот код для создания таблицы на странице:

```
<table>  
  <tbody>  
    <tr>  
      <th>1 столбец</th>  
      <th>2 столбец</th>  
      <th>3 столбец</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>
```

Вот что будет на странице:

**1 столбец 2 столбец 3 столбец**

**4**

**5**

**6**

**7**

**8**

**9**

## Урок 1.9. Создание нумерованных и маркированных списков.

Для добавления изображений на страницу HTML необходимо использовать тег `<ol>` для нумерованных списков и `<ul>` для создания маркированных списков:

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

Вот что будет на странице:

1. 1 элемент
  2. **2 элемент**
  3. 3 элемент
- 1 элемент
  - **2 элемент**
  - 3 элемент

Также можно задавать для нумерованных списков атрибут `start`, соответствующий с какого номера начинается элемент. К примеру:

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

Вот что будет на странице:

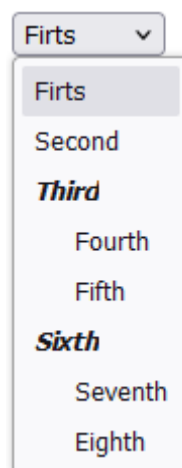
7. 1 элемент
8. 2 элемент
9. 3 элемент

## Урок 1.10. Создание выпадающих списков.

Для добавления изображений на страницу HTML нужно использовать 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>
```

Вот что будет на странице:



## Урок 1.11. Добавление видео и аудио контента.

Для добавления аудио-файлов используем тег `<audio controls>`, а для добавления видео используем тег `<video controls>`. Вот пример в этом коде:

```
<!-- Добавление аудио-файлов -->

<!-- Добавление формата .mp3 -->
<audio controls>
  <source src="audio.mp3" type="audio/mp3">
</audio>

<!-- Продолжение кода на следующей странице -->
```

```

<!-- Добавление формата .wav -->
<audio controls>
  <source src="audio.wav" type="audio/wav">
</audio>

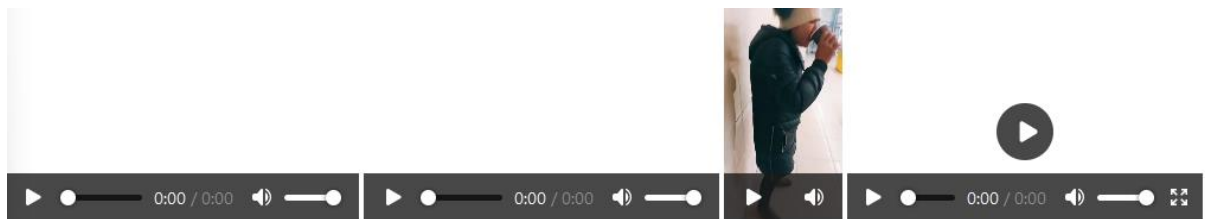
<!-- Добавление видео -->

<!-- Добавление формата .mp4 -->
<videocontrols>
  <source src="video.mp4" type="video/mp4">
</video>

<!-- Добавление формата .avi -->
<videocontrols>
  <source src="video.avi" type="video/avi">
</video>

```

Вот что будет на странице:



Также можно добавлять видео с YouTube-сервиса. Нужно перейти на нужное видео и кликнуть на кнопку «Поделиться» и выбрать «Вставить», где будет тег `<iframe>` с атрибутами и этот код нужно вставить в наш HTML-код.

## Урок 1.12. Использование блоков: `<div>`, `<footer>`, `<header>`, и других.

HTML-элементы `<div>`, `<footer>` и `<header>` широко используются для структурирования и оформления веб-страниц. `<div>` (от "division") – это универсальный контейнер, используемый для группировки других HTML-элементов. Он не имеет особого значения, но часто применяется для организации структуры страницы и применения CSS-стилей. Вот пример в этом коде:

```

<div style="width: 200px;">
  <h1>Header Header Header</h1>
  Main Content
</div>
<!-- Продолжение кода на следующей странице -->

```



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

Вот что будет на странице:

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

Main Content

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

Main Content

Но `<div>` - это единственный контейнер в HTML, потому что есть еще: `<nav>`; `<footer>`; `<header>`; `<section>`; `<article>`; `<aside>`; `<main>`; `<figure>`; `<figcaption>`; `<dialog>`.

- `<nav>`: используется для определения навигационных ссылок. К примеру, меню сайта.

```
<nav>
  <ul>
    <li><a href="#home">Главная</a></li>
    <li><a href="#about">О нас</a></li>
    <li><a href="#contact">Контакты</a></li>
  </ul>
</nav>
```

- `<footer>`: Представляет нижний колонтитул секции или страницы. Часто содержит авторские права, ссылки на политику конфиденциальности и другую информацию.

```
<footer>
  <p>©2024 Компания. Все права защищены.</p>
</footer>
```

- `<header>`: используется для представления ввода или группы навигационных ссылок. Обычно включает логотип, название или навигационные ссылки.

```
<header>
  <h1>Заголовок Сайта</h1>
  <nav>
    <ul>
      <li><a href="#home">Главная</a></li>
      <li><a href="#about">О нас</a></li>
      <li><a href="#contact">Контакты</a></li>
    </ul>
  </nav>
</header>
```

- `<section>`: используется для определения разделов в документе, таких как главы, вкладки и т.д.

```
<section>
  <h2>О нас</h2>
  <p>Здесь можно добавить информацию о вашей компании.</p>
</section>
```

- `<article>`: Представляет автономный фрагмент контента, который может быть распространен независимо. К примеру, блог пост или новостная статья.

```
<article>
  <h2>Статья</h2>
  <p>Текст статьи...</p>
</article>
```

- `<aside>`: Используется для содержимого, косвенно связанного с основным контентом. К примеру, боковая панель со ссылками или рекламой.

```
<aside>
  <h2>Боковая Панель</h2>
  <p>Дополнительный контент или реклама здесь.</p>
</aside>
```

- `<main>`: используется для основного содержимого документа. Должно быть только одно `<main>` на странице.

```
<main>
  <h1>Главный Заголовок</h1>
  <p>Основной контент вашей страницы.</p>
</main>
```

- `<figure>`: Используется для обозначения независимого содержимого, такого как иллюстрация, диаграмма или фото вместе с описанием.

```
<figure>
  
  <figcaption>Подпись к изображению</figcaption>
</figure>
```

- `<figcaption>`: используется для добавления подписи в `<figure>`.

```
<figure>
  
  <figcaption>Подпись к изображению</figcaption>
</figure>
```

- `<dialog>`: используется для создания диалогового окна или модального окна.

```
<dialog>
  <p>Это диалоговое окно.</p>
  <button>Закрыть</button>
</dialog>
```

### Урок 1.13. Выкладка сайта в сеть.

Сервис для загрузки сайта, созданный из HTML-страниц, лучший и бесплатный, это GitHub. Для начала нужно зарегистрироваться. После регистрации в этом сервисе нужно создать публичный репозиторий с файлом README. После создания репозитория, нужно нажать на Add file и Upload files, где будет переход на добавление файлов. После перехода на страницу для загрузки нужно загрузить все файлы, взаимосвязанные с HTML-страницами. Далее нужно перейти на Setting и перейти на , где будет выпадающий список, в котором выбран None. Надо вместо «None» выбрать «main» и нажать кнопку «Save» и подождать некоторое время (примерно, это 5 минут). После ожидания некоторого времени нужно перезагрузить сайт, на котором должна появиться новая ссылка на созданный сайт.

Но это только загрузка сайта в сеть, а если нужно, чтобы пользователь смог найти сайт через поисковую систему (), то нужно еще подключить страницу к поисковой системе. Вот справки, как добавить ссылку на сайт в поисковую систему Google (Ссылка: <https://support.google.com/webmasters/>) и для Bing (Ссылки: <https://www.bing.com/webmasters/help>).

Это не все HTML-теги, которые были показаны в этой надбавке, но это базовые теги, которые нужно знать. Все теги доступны на сайте [css.in.ua](https://css.in.ua), где есть все теги, скрипты JavaScript и CSS компоненты для стилизации тег. Там также показаны все атрибуты для каждого тэга. Вот ссылка на сайт: <https://css.in.ua/html/tags>. Также там показано, какие теги можно использовать только в версии HTML 5.0, которые можно использовать во всех версиях HTML (кроме 5.0, потому что они в этой версии не поддерживаются) и которые поддерживаются на всех версиях HTML.

## II РАЗДЕЛ. СТИЛИЗАЦИЯ СТРАНИЦ.

### Урок 2.1. Что такое CSS.

Если HTML служит для размещения элементов на странице, CSS (Cascade Style Sheet) отвечает за их стилизацию. Язык CSS тоже не язык программирования.

### Урок 2.2. Использование CSS в HTML.

Для использования стилизации в HTML-страницах можно производить 3 путями:

1. **Встроенный стиль (Использование атрибута style):** в атрибуте style можно вписывать настройки для тега. К примеру:

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

2. **Встроенная таблица стилей (Использование тега <style>):** в теге <style> можно вписывать селекторы и их опции для тег, id и классов в странице. Он может быть в тезисе <body> или <head>. К примеру:

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

3.

4. **Внешняя таблица стилей (импорт файла ".css"):** используем тег <link>. К примеру:

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

Задание стиля состоит из:

```
селектор {
  опция: значение;
}
```

Также можно делать значения через переменные:

```
селектор {  
  --изменение: значение  
опция:var(--изменение);  
опцияДва:var(--изменение);  
  
  /* При этом в двух опциях будет одинаковое значение */  
}
```

Кстати говоря! Где в коде: «/\* При этом в двух опциях будет одинаковое значение \*/», эта строка есть – комментарий.

Пусть мы делаем стилизацию для этого кода:

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

Теперь нужно сделать, чтобы наш тег <p> имел синий шрифт на 14 пикселей. Тогда нужно нам говорить селектор «p», потому что это нужно сделать для тега <p>. Вот таким образом мы будем делать в коде style.css:

```
p{  
  /* Задание цвета */  
  color:blue;  
  
  /* Задание размера шрифта в пикселях */  
  font-size:14px;  
}
```

Теперь на странице увидим:

Hello

Также можно сделать через переменные, способ который тоже сделает такой же результат, как первый:

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

Можно также не указывать название цвета в значении, а использовать функцию RGB.

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

  /* Или таким образом */
  color: #XXYYZZ;
  /*
  XX, xxx – красный оттенок
  YY, yyy – Зеленый оттенок
  ZZ, zzz – синий оттенок
  */
}
```

Также есть понимание как псевдо-классы. Вот их использование в таблице стиля:

```
селектор: псевдо-класс {
  опция: значение;
}
```

Псевдо-класс – состояние селектора.

Селектор – это в целом теги, id (#id) и классы (.class).

Вот таблица всех базовых псевдоклассов:

Псевдо-класс	Условие
hover	Когда пользователь нажал курсором на селектор левой кнопкой мышки
focus	Когда пользователь нажал курсором на селектор правой кнопкой мышки
active	Когда пользователь нажал и не отпускает левую кнопку мышки на селектор
visited	Когда пользователь нажал на этот селектор и перешел на страницу.

Вот пример использования псевдо-классов:

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

## Урок 2.3. Стилизация классов и ID.

Есть код, имеющий теги: <h1>; <h2>; <p>, имеющие класс cls и id i. Вот и он:

```
<!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>
```

Теперь нужно сделать так, чтобы все текстовые тэги были красными.

Можно сделать так:

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

Можно и таким образом, но если они имеют общий класс или id, то можно сделать так:

```
/* Для классов */
.cls{
    color:red;
}

/* Для id */
#i{
    color:red;
}
```



## Урок 2.4. Общие настройки.

Вот код с общими опциями в CSS:

```
селектор {  
    /* Основы стилизации элементов */  
  
    /* Задаёт цвет текста */  
    color:red;  
  
    /* Определяет семейство шрифта */  
    font-family:'Segoe UI';  
  
    /* Размер шрифта (В пикселях) */  
    font-size:red;  
  
    /* Толщина шрифта (Число) */  
    font-weight:100;/* 100 – тонкий, наименьшая толщина */  
    font-weight:400;/* 400 – Стандартный */  
    font-weight:600;/* 600 – жирный */  
  
    /* Стил шрифта (italic, normal или bold) */  
    font-style:italic;/* Наклонный стил шрифта */  
    font-style:normal;/* Обычный стил шрифта */  
    font-style:bold;/* Жирный стил шрифта */  
  
    /* Отделка текста (подчеркивание, перечеркивание) */  
    text-align:red;  
  
    /* Отделка текста (подчеркивание, перечеркивание) */  
    text-decoration:underline;/* подчеркивание */  
    text-decoration:overline;/* перечеркивание */  
  
    /* Изменение регистра букв */  
    text-transform:uppercase;/* Все буквы станут прописными */  
    text-transform:capitalize;/* Первая буква каждого слова будет  
                             большой */  
    text-transform:lowercase;/* Все буквы станут маленькими */  
    text-transform:none;/* Оставляет текст без изменений, как есть  
                        за по умолчанию */  
    text-transform:inherit;/* Наследует свойство от родительского  
                           элемента*/  
    text-transform:initial;/* Значение по умолчанию, которое  
                           определенное браузером */  
    text-transform:unset;/* Сбрасывает свойство к значению, которое  
                           зависит от контекста */  
  
    /* Продолжение кода на следующей странице */
```

```
/* Высота строки текста (В пикселях) */
line-height:1.5px;

/* Интервал между символами (В пикселях) */
letter-spacing:0.5px;

/* Интервал между словами (В пикселях) */
word-spacing:2px;

/* Прозрачность */
opacity:0.7;/* 70% прозрачность */

/* Уровень размытия */
filter:blur(5px);/* Размытие на 5 пикселей*/

/* Основы макетирования */

/* Определяет тип блока для HTML-элемента
(block, inline, flex, grid)*/
display:block;/* Блочный элемент */
display:inline;/* строчный элемент */
display:flex;/* Гибкий контейнер*/
display:grid;/* Сетевой контейнер*/

/* Устанавливает ширину элемента (В пикселях) */
width:10px;

/* Устанавливает высоту элемента (В пикселях) */
height:10px;

/* Наружные отступы вокруг элемента (В пикселях) */
margin:10px;

/*Внутренние отступы вокруг содержимого элемента (В пикселях)*/
padding:10px;

/* Устанавливает границу элемента (ширина, стиль, цвет) */
/* Цвет */
border:red;
/* Ширина (В пикселях) */
border:10px;
/* Стиль */
border:none;/* Отсутствие границы */
border:currentColor;/* Цвет границы равен цвета текста */
border:dashed;/* Граница состоит из коротких штрихов*/
border:dotted;/* Граница состоит из точек */
/* Продолжение кода на следующей странице */
```

```
border:double;/* Двойная линия границы */
border:groove;/* Выглядит как вырезанный в фоне */
border:hidden;/* Граница скрыта (эквивалентно 'none') */
border:inherit;/* Наследует стиль границы */
border:inset;/* Выглядит как утопленный в фоне */
border:initial;/* Инициализация к значению за по умолчанию */
border:medium;/* Средний размер границы (эквивалентно 3px по
                умолчанию) */
border:outset;/* Выглядит как выпуклый над фоном */
border:ridge;/* Выглядит как гребень на фоне */
border:solid;/* Прямая непрерывная линия границы */
border:thick;/* Толстая граница */
border:thin;/* Более тонкая граница */
border:unset;/* Сброс до значения по умолчанию */

/* Отступы от краев родительского элемента для
позиционированных элементов (В пикселях) */
top:10px;/* Отступление от верхней части */
right:10px;/* Отступление от правой части */
bottom:10px;/* Отступление от нижней части */
left:10px;/* Отступление от левой части */

/* Позиция селектора */
position:absolute;/* Элемент позиционируется относительно
ближайшего предка с non-static*/
position:fixed;/* Элемент остается на одном месте
при прокрутке */
position:relative;/* Элемент позиционируется относительно
его обычного места*/
position:static;/* Элемент размещается в стандартном
потоке документа */
position:sticky;/* Элемент позиционируется относительно
прокрутки */
position:inherit;/* Элемент наследует позицию от
родительского элемента*/
position:initial;/* Элемент имеет значение по умолчанию
(static) */
position:unset;/* Сброс к значению, которое зависит от
контекста */

/* Продолжение кода на следующей странице */
```

```
/* Основы работы с фоном */

/* Задаёт цвет текста */
background:red; /* Там можно выбрать
изображение или цвет */

/* Цвет фона */
background-color:blue;

/* Фоновое изображение */
background-image:url('image.png');

/* Способ повторения фонового изображения */
background-repeat:no-repeat; /* без повторения */
background-repeat:inherit; /* Наследовать */
background-repeat:initial; /* Инициализирует */
background-repeat:repeat; /* Повторяется по всей площади */
background-repeat:repeat-x; /* Повторяется только по
горизонтали */
background-repeat:repeat-y; /* Повторяется только по
вертикали */
background-repeat:round; /* Растягивается, чтобы поместиться
в контейнер */
background-repeat:space; /* Размещается с пробелами между
повторениями */
background-repeat:unset; /* Сброс */

/* Позиция фонового изображения */
background-position:top left; /* Устанавливает позицию
изображение в верхний левый угол */
background-position:top center; /* Позиционирует изображение в
верхний центр */
background-position:top right; /* Позиционирует изображение в
верхний правый угол */
background-position:center left; /* Позиционирует изображение в
центре слева */
background-position:center center; /* Центрирующее изображение в
середине элемента */
background-position:center right; /* Позиционирует изображение в
центре справа */
background-position:bottom left; /* Позиционирует изображение в
нижний левый угол */
background-position:bottom center; /* Позиционирует изображение в
нижний центр */
background-position:bottom right; /* Позиционирует изображение в
нижний правый угол */

/* Продолжение кода на следующей странице */
```

```
background-position:10px 20px;/* Позиционирует изображение на
расстояния 10px слева и 20px
сверху*/
/* Значение по умолчанию: top left */

/* Размер фонового изображения */
background-size:auto;
background-size:cover;
background-size:contain;
background-size:100px 200px;/* Устанавливает ширину и
высоту изображения.
100px – ширина,
200px высота */
background-size:50% 40%;/* Устанавливает ширину и высоту
как процент от размера
элемента */

/* Основы работы с текстом */

/* Горизонтальное выравнивание текста */
text-align:left;/* Выравнивает текст по левому краю */
text-align:center;/* Выравнивает текст по центру */
text-align:right;/* Выравнивает текст по правому краю */
text-align:justify;/* Выравнивает текст по обоим краям,
создавая равномерный интервал */

/* Тень текста */
text-shadow:none;/* Отсутствие тени */

/* Размер тени (X, Y, Blur, Color)
X – расположение от левой части селектора
Y – расположение от верхней части селектора
Blur – Размытие
Color – Цвет тени*/
text-shadow:2px 2px 4px rgba(0,0,0,0.5);

/* Обрезка текста, выходящего за пределы элемента */
text-overflow:clip;/* Обрезает выходящий текст
пределы контейнера */
text-overflow:ellipsis;/* Отображает три точки "..."
в конце переполненного текста */

/* Как обрабатываются пробелы и переносы строк */
white-space:normal;/* Обрезает пробелы и переносы
строк по умолчанию */
white-space:nowrap;/* Не позволяет переносить строки,
текст продолжается в одну строку */
/* Продолжение кода на следующей странице */
```

```
white-space:pre;/* Сохраняет все пробелы и переносы строк */

white-space:pre-wrap;/* Сохраняет пробелы,
но разрешает переносы строк */
white-space:pre-line;/* Сохраняет пробелы, но удаляет лишние
пробелы и хранение переносов строк */

/* Основы работы с коробочной моделью */

/* Алгоритм расчета ширины и высоты элемента */
box-sizing:content-box;/* Рассчитывает ширину и высоту
только для контента */
box-sizing:border-box;/* Включает ширину и высоту
границ и отступлений в
общий размер */

/* Поведение переполненного элемента
(visible, hidden, scroll, auto) */
overflow:visible;/* Переполненный контент видимый
за пределами элемента */
overflow:hidden;/* Переполненный контент обрезаются
и не видно*/
overflow:scroll;/* Добавляет прокрутку, если контент
выходит за пределы элемента */
overflow:auto;/* Добавляет прокрутку только когда
это необходимо */

/* Использование градиента */
/* linear-gradient(to XXX, от_цвет, до_цвет)
XXX – направление, куда должен идти градиент:
right – вправо;
left – влево;
top – вверх;
bottom – вниз;
right – вправо;
right – вправо; */
background:linear-gradient(to top,red,yellow);
background:linear-gradient(yellow,red,yellow,blue,green);
/*
to top - направление (направления всегда стоят в начале,
а дальше цвета)

yellow, red, yellow, blue, green - набор цветов,
которые должны быть в градиенте */
}
```

## Урок 2.5. Display-опция.

Вот коды с общими опциями в CSS:

```
селектор {  
  display:block;/* Стоит в контейнере и  
  ширина селектора зависит от  
  ширины контейнера в котором  
  находится этот селектор */  
}
```

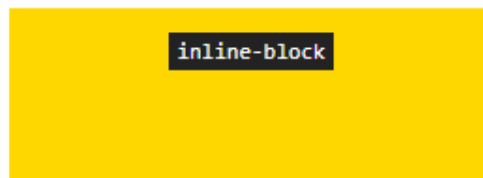
В браузере будет:



Следующий код:

```
селектор {  
  display:inline-block;/* Стоит в контейнере и  
  ширина селектора зависит от  
  ширины селектора */  
}
```

В браузере будет:



Следующий код:

```
селектор {  
  display:none;/* Селектор будет скрыт */  
}
```

В браузере будет:



Свойство `display` может получать 20 значений:

`block`- элемент показывается как блочный. Указав это значение для элемента `<span>`- он начнет вести себя как блочный, то есть его содержимое будет всегда начинаться с новой строки.

`inline`- элемент отображается как строчный (`<span>`). Значение `inline` отменяет особенность блочных элементов всегда начинаться с новой строки.

`inline-block`- это значение создает блочный элемент, который ведет себя подобно строчному элементу. Его внутренняя часть форматируется как у блочного элемента, а сам элемент – как строчный. Так себя ведет элемент `<img>`.

`inline-table`- определяет, что элемент является таблицей `<table>`, но при этом таблица ведет себя как строчный элемент и обтекается другими элементами.

`inline-flex`- Элемент ведет себя как строчный и излагает содержимое согласно флекс-модели. Появился только у CSS3.

`flex`- Элемент ведет себя как блочный и выкладывает содержимое согласно флекс-модели

`list-item`- элемент ведет себя так же как элемент `<li>`.

`none`- скрывает элемент. Макет формируется, словно элемента и не было. Сделать видимым элемент можно с помощью скриптов. При этом происходит переформатирование данных на странице с учетом вновь добавленного элемента.

`run-in`- устанавливает элемент как блочный или строчный в зависимости от контекста.

`table`- придает элементу элемента качеств таблицы `<table>`.

`table-caption`- задает заголовок таблицы подобно применению `<caption>`

`table-cell`- указывает, что элемент представляет собой элемент таблицы (`<td>` или `<th>`)

`table-column-group`- Элемент ведет себя как он элемент `<colgroup>`.

`table-column`- Элемент ведет себя как он элемент `<col>`.

`table-footer-group`- используется для хранения одной или нескольких строк ячеек, которые отображаются в самом низу таблицы. По своему действию схож с работой `<tfoot>`

`table-header-group`- Элемент предназначен для хранения одной или нескольких строк ячеек, представленных в верхней части страницы. По своему действию схож с работой `<thead>`

`table-row`- элемент отображается как строка таблицы `<tr>`



`table-row-group`- элемент аналогичен действию элемента `<tbody>`.

[initial](#)- устанавливает свойство в значение без задания

[inherit](#)- указывает на наследственность свойства от своего родительского элемента

**Значение без задания:** *inline*

**Следует:** Нет

**Аниммируется:** Да

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

## Урок 2.6. Фильтры.

CSS свойство `filter` определяет, какие визуальные эффекты применить к элементу, например размытие или контраст. Чаще применяется к элементу `<img>`).

Свойство `filter` может получать 14 значений:

`none`- эффекты отсутствуют. Без задания.

`blur(px)` - задает размытие. Чем больше значение, тем больше размытие. Если значение не указано, используется 0. Отрицательное значение запрещено.

`brightness (%)` - яркость изображения:

- 0% сделает изображение полностью черным.
- 100% (1) Изображение останется без изменений. Без задания.
- Значение более 100% сделает изображение светлее.

`contrast (%)` - контрастность изображения:

- Значение менее 100% или 1 уменьшит контрастность.
- 100% или 1 Изображение останется без изменений. Без задания.
- Значение более 100% или 1 увеличит контрастность.

Отрицательное значение не допускается. Пустое значение рассматривается как 1.

`grayscale (%)` - Преобразует изображение в черно-белое.

- 100% или 1 - преобразует изображение в черно-белое.
- 0% или 0- Изображение не меняется.
- От 0% до 100% (или от 0 до 1) изменяют цветовую гамму картинки.

Отрицательные значения запрещены. Пустое значение рассматривается как 0.

`hue-rotate (deg)` - изменяет цветность изображения за счет поворота оттенка на цветовом круге.

- 0 или 360 градусов оставляет изображение неизменным.
- Любые другие значения будут изменять цветовую гамму.

Отрицательные значения разрешены. Максимальное значение 360deg. Пустое значение воспринимается как 0deg.

`invert (%)` - Инвертируйте цвета в изображении. По своему действию похоже на превращение фотографии в негатив.

- 0% или 0 – Изображение будет без изменений.
- 100% или 1 – полностью инвертирует цвета картинки.
- От 0% до 100% или от 0 до 1 – частично инвертирует цвета.

Отрицательные значения запрещены. Пустое значение рассматривается как 0.

`opacity (%)` - задает степень прозрачности изображений. По своему действию похожа на свойство `opacity`, но некоторые браузеры для фильтров применяют аппаратное ускорение, чтобы повысить их производительность.

- 100% или 1 Изображение останется без изменений.
- 0% или 0 Изображение станет полностью прозрачным.
- 0% до 100% или от 0 до 1 задает степень прозрачности изображения.

Отрицательное значение не допускается. Пустое значение рассматривается как 1.

`saturate (%)` - насыщенность цветов в изображении.

- 0% или 0 убирает насыщенность цвета в изображении, превращая его в черно-белое.
- 100% или 1 оставляет изображение неизменным.
- 100% и более делает изображения более насыщенными.

Отрицательное значение не допускается. Пустое значение рассматривается как 1.

`sepia (%)` - превращает изображение в сепию – так называемое черно-белое изображение с коричневым оттенком. Сепия придает фотографиям старинный вид:

- 0% или 0 Изображение не меняется.
- 100% или 1 сепия в максимальную силу.
- 0% до 100% или от 0 до 1 линейно применяют сепию.

Отрицательные значения не допускаются. Пустое значение рассматривается как 0.

`url ()` - принимает расположение XML файла, определяющего SVG фильтр. Ссылка также может включать в себя якорь для конкретного фильтра. Пример: `filter: url(svg-url#element-id)`

[initial](#)- устанавливает свойство в значение без задания  
[inherit](#)- указывает на наследственность свойства от своего  
родительского элемента

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

Применяется эффект тени к изображению.

Возможные значения:

- `h-shadow`- Обязательно. Задаёт значение ширины горизонтальной тени. Отрицательные значения отобразят тень слева от изображения.
- `v-shadow`- Обязательно. Установка пикселя для вертикальной тени. Отрицательные значения отобразят тень над изображением.
- `blur`- необязательно. Это третье значение, и оно должно быть в пикселях. Придаёт эффект размытия тени. Большее значение

будет создавать большее размытие (тень становится больше и светлее). Отрицательные значения не допускаются. Если значение не указано, используется 0 (тени не будут видны).

- `spread` – необязательно. Это четвертое значение, и оно должно быть в пикселях. Положительное значение заставит тень расширяться, а отрицательные значения заставят тень сокращаться. Если это значение не указано, то оно будет 0 (тень будет иметь тот же размер, что и элемент). Примечание: Chrome, Safari, Opera возможно и другие браузеры, не поддерживающие это значение.
- `цвет` – необязательно. Добавляет цвет тени. Если не указано, то цвет зависит от браузера (чаще всего тень будет черного цвета).

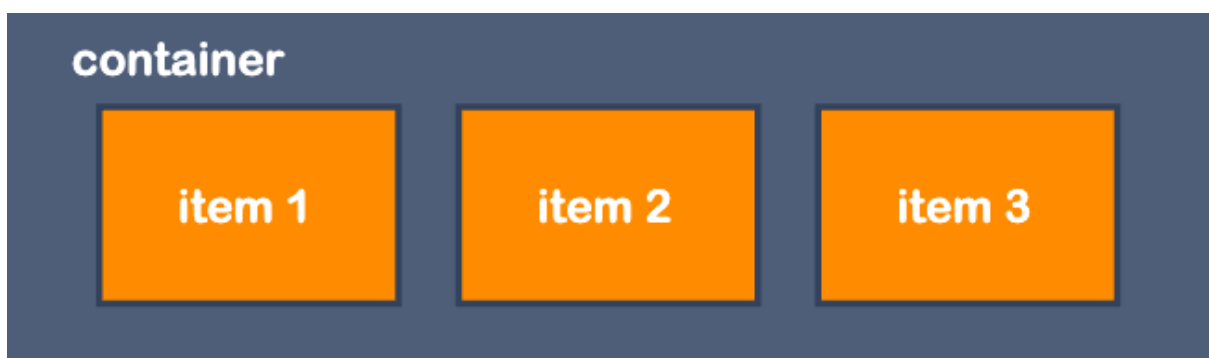
Пример создания красной тени, ширина и высота которой 8px с эффектом размывания в 10 пикселей:

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

Подсказка: Этот фильтр аналогичен свойству `box-shadow`.

## Урок 2.7. Flexbox.

**Flexbox**, сокращенно от Flexible Box Layout, – это мощный модуль верстки в CSS, предназначенный для оптимизации выравнивания, распределения и организации элементов внутри контейнера.



Модель компоновки flexbox работает на основе связи "отец-ребенок". Отец, который называется flex-контейнер, отвечает за диктовку макета своим дочерним элементам, известным как flex-элементы. Используя ряд свойств flexbox, мы можем точно контролировать размер, позицию и порядок расположения flex-элементов в flex-контейнере.

Примечание! Внутри flex-контейнера многие обычные правила позиционирования не применяются так, как ожидалось. Давайте рассмотрим следующий список:

- Работа с flex-элементами отличается от работы с элементами inline или block-level.
- В отличие от элементов block-level, flex-элементы не перекрывают другие элементы и не накладываются друг на друга.
- Поля, применяемые на краях родительского элемента, не выходят за его пределы.

## flex модель

Поскольку мы не имеем дело со встроенными или блочными элементами, поток документов во флекс-модели определяется осями флекс-контейнера, по которым выравниваются все элементы. Чтобы инициировать поведение flexbox, мы используем свойство display: flex; к родительскому элементу, содержащему все элементы.

Мы можем влиять на направление расположения flex-элементов. По умолчанию, flex-элементы размещаются в ряд. Однако мы можем изменить это поведение с помощью flex-direction. Давайте сравним документооборот без флекса и с флексом:

Код CSS:

```
/* Apply flex to the parent element.
/* В этом положении, это элемент <ul> с "list" class name.
*/
.list{
  display:flex;
}

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

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

Код 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>
      <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 странице:

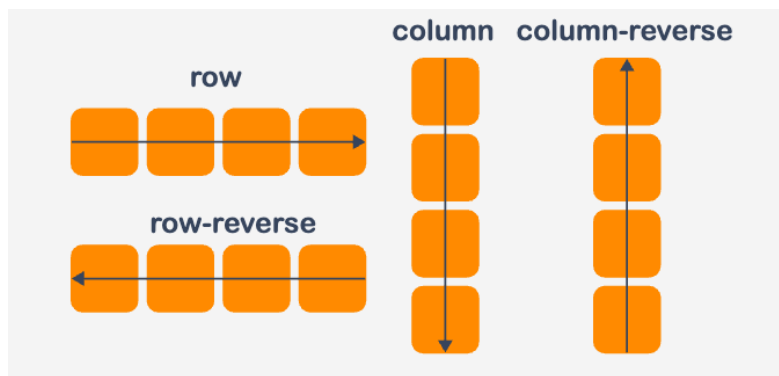
#### Default

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

#### Flex

Свойство `flex-direction` позволяет определить направление, в котором расположены flex-элементы. По умолчанию это свойство имеет значение `row`, что приводит к компоновке слева направо в браузерах, использующих английский по умолчанию. Вот значения, которые можно вставить. `flex-direction: row | column | row-reverse | column-reverse;`

Вот как располагаются элементы с использованием flex-direction:



Рассмотрим для примера следующую навигацию по сайту

Вот 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 это CSS-код:

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

.navigation-item{
  padding:10px;
  border-radius:6px;
  background-color:violet;
}

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

.navigation-link{
  color:wheat;
}

ul{
  list-style:none;
}

a{
  text-decoration:none;
}
```

Где было XXX, там будем указывать значение. Вот таблица, когда XXX равна какому-либо значению:

Когда XXX равно:	Результат
row	
row-reverse	



column	
column-reverse	

## align-items

Свойство align-items в CSS используется для выравнивания элементов по вертикали в контейнере с display:flex или display:grid. Для таблиц это свойство также не применяется непосредственно, но мы можем использовать Flexbox или Grid внутри ячеек таблицы для достижения нужного выравнивания: align-items: stretch | центр | flex-start | flex-end | baseline;

Вот как располагаются элементы с использованием align-items:

**stretch**



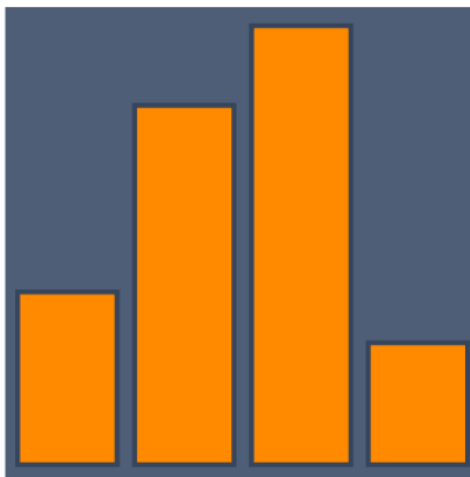
**center**



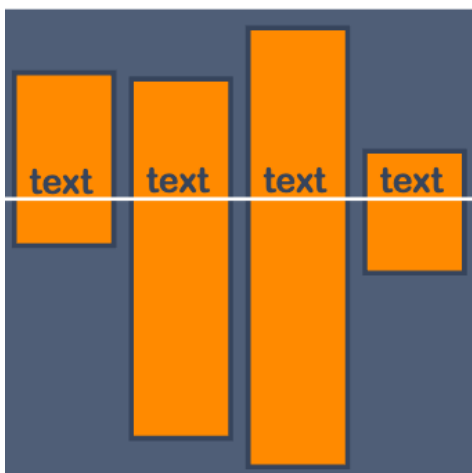
**flex-start**



**flex-end**



**baseline**



Теперь сделаем коды:

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;
}
```

Где было XXX, там будем указывать значение. Вот таблица, когда XXX равна какому-либо значению:

Когда XXX равно:	Результат				
stretch	 Lorem, ipsum dolor.	 Lorem ipsum dolor sit amet consectetur adipiscing elit.	 Lorem ipsum dolor sit amet.	 Lorem ipsum dolor sit, amet consectetur adipiscing elit.	 Lorem ipsum dolor sit amet consectetur adipiscing elit.

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. Анимация.

Иногда нам приходится создавать анимацию для изменения элементов на веб-странице с контролирующими факторами, такими как скорость, время задержки и продолжительность. В таких случаях для выполнения этой задачи может быть использовано свойство "transition". Элемент всегда имеет два состояния: начальное и конечное. Мы можем управлять поведением изменения элемента с помощью следующих свойств:

```
.transitionClass{
  /* Имя свойства CSS, которое нам
  нужно анимировать (значение должно
  иметь название той опции, которая должна
  измениться при изменении статуса
  селектора) */ /* Продолжение кода на следующей странице */
```

```

transition-property:color;

/* Время, в течение которого нам нужно
изменить состояние элемента. Он указывается
в секундах (s) или миллисекундах (ms) */
transition-duration:300ms;
/* Продолжение кода на следующей странице */
/* Он задает кривую скорости эффекта
перехода. Типичными значениями являются:
ease; linear; ease-in; ease-out; ease-in-out.*/
transition-timing-function:ease;

/* Задержка времени до начала эффекта перехода.
Он указывается в секундах (s) или
миллисекундах (ms). */
transition-delay:2s;

/* 1 секунда = 1000 миллисекунда */
}

```

Вот пример страницы, в которой есть контейнер (Тег <div>), при приближении курсора мышки к нему изменится его цвет:

```

<!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>

```

Если нужно изменять значения всех опций, нужно указывать:

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

В CSS свойство transition-timing-function определяет, как скорость анимации изменяется в течение времени. Вот объяснение основных значений:

**1. ease:**

- Это значение по умолчанию.
- Скорость анимации сначала растет, затем замедляется.
- Создает эффект ускорения в начале и замедления в конце.
- График скорости имеет S-образную форму.

**2. linear:**

- Скорость анимации постоянно на протяжении всего времени.
- Нет ускорения или замедления.
- График скорости представляет собой прямую линию.

**3. ease-in:**

- Скорость анимации растет в начале, затем добивается постоянного значения.
- Создает эффект ускорения в начале.
- График скорости начинается с нуля и ускоряется.

**4. ease-out:**

- Скорость анимации сначала постоянна, затем замедляется до конца.
- Создает эффект замедления в конце.
- График скорости начинается с скорости и замедляется до конца.

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

- Скорость анимации сначала растет, затем замедляется.
- Создает эффект ускорения в начале и замедления в конце.
- График скорости имеет S-образную форму, но более выражен, чем у ease.

Также можно писать просто transition в таблице стилей:

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

Есть также анимация, которая дает команду селектору крутиться по часовой стрелке:

```
/* Сделать это 5 раз */  
animation-iteration-count:5;  
  
/* Сделать это бесконечно*/  
animation-iteration-count:infinite;
```

Также есть опция animation-direction, которая применяется следующим образом:

```
/* элемент перемещается слева направо,  
повторяется снова с начала. */  
animation-direction:normal;  
  
/* элемент перемещается справа налево,  
повторяется снова с конца */  
animation-direction:reverse;  
  
/* элемент перемещается слева направо,  
потом справа налево, и так далее*/  
animation-direction:alternate;  
  
/* элемент перемещается справа налево,  
затем слева направо, и так далее*/  
animation-direction:alternate-reverse;
```

## Урок 2.9. Адаптация.

Адаптация сайта может включать в себя различные аспекты, такие как дизайн, функциональность, производительность и соответствие современным стандартам. Адаптивный дизайн обеспечивает корректное отображение сайта на разных устройствах, от мобильных телефонов до настольных компьютеров. Оптимизация скорости загрузки сайта включает в себя использование кэширования, уменьшение размера изображений, минимизацию CSS и JavaScript файлов. Обеспечение доступности сайта для людей с ограниченными возможностями включает использование



соответствующих атрибутов ARIA, обеспечение высокой контрастности текста и поддержка навигации с клавиатуры. Адаптация сайта для поддержки различных языков включает в себя использование фреймворков для локализации и перевода контента.

Вот синтаксис при использовании адаптации:

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

1. <media\_type> – декларирует тип устройства. Возможные значения:
  - all – значение по умолчанию. Если ничего не указано, правило носителя работает для всех устройств;
  - print – правило носителя работает для устройств, производящих печатные документы, например, принтеров;
  - screen – правило носителя работает для устройств, производящих печатные документы, например, принтеров.
2. <media\_feature> – заявляет характеристики устройства. Наиболее распространенное применение:
  - min-width: – минимальная ширина окна просмотра;
  - max-width: – максимальная ширина окна просмотра.
3. <operator> - media-type и media-feature отделяются необязательным логическим оператором. Его значения могут быть: and или or.

```
/* 1-ый пример */  
@media screen and(max-width:1200px) {  
    .container{  
        color:aliceblue;  
    }  
}  
  
/* 2-ой пример */  
@media screen and(min-width:1680px) {  
    .container{  
        color:gainsboro;  
    }  
}  
  
/* В первом примере мы сказали браузеру,  
что для любых устройств с шириной экрана
```

менее 1200 пикселей или равно 1200 пикселей  
применить свойство `color` со значением  
`aliceblue` к элементу с контейнером имени класса.  
(Продолжение кода на следующей странице)  
Во втором примере мы сообщили браузеру,  
что если ширина экрана любого устройства превышает  
1680 пикселей или равно 1680 пикселей, свойство  
цвета со значением `gainsboro` должно быть применено  
к элементу под названием класса контейнера. `*/`

## III РАЗДЕЛ. Скрипты.

### Урок 3.1. Что такое JavaScript.

JavaScript (JS) – динамический, объектно-ориентированный прототипный язык программирования. Реализация ECMAScript. Чаще всего используется для создания сценариев веб-страниц, что позволяет на стороне клиента (устройства конечного пользователя) взаимодействовать с пользователем, управлять браузером, асинхронно обмениваться данными с сервером, изменять структуру и внешний вид вебстраницы.

JavaScript классифицируют как прототипный (подмножество объектно-ориентированного), скриптовый язык программирования с динамической типизацией. Кроме прототипной, JavaScript также частично поддерживает другие парадигмы программирования (императивную и частично функциональную) и некоторые соответствующие архитектурные свойства, в частности динамическая и слабая типизация, автоматическое управление памятью, прототипное подражание, функции как объекты первого класса.

Если мы изучали языки HTML и CSS, не относящиеся к языкам программирования, JavaScript относится к языкам программирования, а HTML и CSS относятся к языкам, которые строят графику, так как они не могут выполнять циклы и на этих языках невозможно сделать функции, которое может сделать только язык программирования, например JavaScript.

### Урок 3.2. Добавление и исполнение скриптов на веб-странице.

Чтобы добавить JS-скрипты, можно сделать 3 способами:

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

    <!-- 2 способ - добавление ссылки на
    скрипт к тегу <script> -->
    <script src="путь/к/скрипта.js"></script>
    <!--Продолжение кода находится на следующей странице -->
```

```
<!-- 3 способ - добавление ссылки на
скрипт к тегу <script> из интернета -->
<script
type="application/javascript"
src="http://страница.co.скриптом/"
></script>
</body>
</html>
```

### Урок 3.3. События.

События – атрибуты в тэгах, которые выполняют скрипт при изменении состояния тэга. Вот код с базовыми атрибутами событий, которые нужно знать:

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

<!-- Вызывается после загрузки элемента завершено -->
<a onload="console.log('Hello World');"></a>

<!-- Вызывается, когда страница разгружена, или окно браузера
было закрыто. -->
<a onunload="console.log('Hello World');"></a>

<!-- Скрипт выполняется, когда элемент теряет фокус. -->
<a onblur="console.log('Hello World');"></a>

<!-- Вызывается в момент, когда элемент получает фокус. -->
<a onfocus="console.log('Hello World');"></a>

<!-- Вызывается, когда пользователь что-то пишет в поле поиска
(для <input type="search">) -->
<a onsearch="console.log('Hello World');"></a>

<!-- вызывается после того как любой текст был выбран в
элементе. -->
<a onselect="console.log('Hello World');"></a>

<!-- Вызывается при отправке формы -->
<a onsubmit="console.log('Hello World');"></a>

<!-- Событие вызывается, когда пользователь зажал
(нажимает и не отпускает) клавишу. -->
<a onkeydown="console.log('Hello World');"></a>
```

```
<!--Продолжение кода находится на следующей странице -->
<!-- Вызывается, когда пользователь нажал на клавишу -->
<a onkeypress="console.log('Hello World');"></a>

<!-- Вызывается, когда пользователь отпускает клавишу -->
<a onkeyup="console.log('Hello World');"></a>

<!-- Возникает при двойном щелчке левой кнопкой
мышки на элементе -->
<a ondblclick="console.log('Hello World');"></a>

<!-- вызывается, когда пользователь зажимает левой кнопкой
мышки на элементе. -->
<a onmousedown="console.log('Hello World');"></a>

<!-- Вызывается, когда курсор мыши перемещается над элементом -
->
<a onmousemove="console.log('Hello World');"></a>

<!-- Вызывается, когда курсор выходит за пределы элемента -->
<a onmouseout="console.log('Hello World');"></a>

<!-- Выполняется, когда курсор наводится на элемент -->
<a onmouseover="console.log('Hello World');"></a>

<!-- Вызывается, когда пользователь отпускает кнопку мыши -->
<a onmouseup="console.log('Hello World');"></a>

<!-- Вызывается, когда пользователь копирует содержимое
элемента -->
<a oncopy="console.log('Hello World');"></a>

<!-- Вызывается, когда пользователь вырезает содержимое
элемента -->
<a oncut="console.log('Hello World');"></a>

<!-- Вызывается, когда пользователь вставляет содержимое в
элемент -->
<a onpaste="console.log('Hello World');"></a>

<!-- Выполняется при прерывании какого-либо события -->
<a onabort="console.log('Hello World');"></a>
```

Это те события, которые нужно знать. Они работают на всех версиях HTML. Но есть события в HTML, которые работают только на HTML 5.0.

Таблица с событиями, работающими только в HTML 5.0, есть на следующей странице.

События	Состояние тега
onplay	Скрипт вызывается когда медиа данные готовы начать воспроизведение.
onafterprint	Скрипт выполняется только после того, как документ напечатан.
onbeforeprint	Скрипт выполняется перед тем, как документ напечатан.
onbeforeunload	Скрипт выполняется когда документ вот-вот будет выгружен
onhashchange	Скрипт выполняется когда там произошли изменения части якоря в URL
onmessage	Скрипт выполняется, когда вызвано сообщение.
onoffline	Срабатывает, когда браузер начинает работать в автономном режиме
ononline	Срабатывает, когда браузер начинает работать в режиме онлайн.
onpagehide	Скрипт выполняется когда пользователь переходит на другую страницу.
onpageshow	Скрипт выполняется, когда пользователь заходит на страницу.
onpopstate	Скрипт выполняется при изменении истории одного окна.
onresize	Скрипт выполняется при изменении размера окна браузера.
onstorage	Скрипт выполняется при обновлении содержимого Web Storage.
oncontextmenu	Скрипт выполняется, когда вызывается контекстное меню.
oninput	Скрипт вызывается когда пользователь вводит данные в поле.
oninvalid	Скрипт выполняется, когда элемент недействителен.
onreset	Вызывается при нажатии в форме кнопки типа Reset.
ondrag	Периодически вызывается при операции перетаскивания.
ondragend	Вызывается, когда пользователь отпускает перемещаемый элемент.

ondragenter	Вызывается, когда перетаскиваемый элемент входит в целевую зону.
ondragleave	Вызывается, когда перетаскиваемый элемент выходит из зоны назначения.
ondragover	Вызывается, когда перетаскиваемый элемент находится в зоне назначения.
ondragstart	Вызывается, когда пользователь начинает перетаскивать элемент или выделенный текст.
ondrop	Вызывается, когда перетаскиваемый элемент падает в зону назначения.
onscroll	Вызывается при прокрутке содержимого элемента (или веб-страницы).
onwheel	Вызывается, когда пользователь прокручивает колесо мыши.
oncanplay	Скрипт выполняется когда файл готов, для начала воспроизведения (когда он буферизирован достаточно, чтобы начать воспроизведение)
oncanplaythrough	Скрипт выполняется, когда контент может быть воспроизведен без прерывания на буферизацию.
oncuechange	Скрипт выполняется когда изменяется кий в <track> элемента
ondurationchange	Вызывается при изменении длины медиа файла.
onemptied	Вызывается, когда доступ к медиа контенту обрывается (пропало соединение с сетью).
onended	Вызывается, когда медиа элемент полностью воспроизвел свое содержание.
onshow	Вызывается, когда элемент <menu> отображается как контекстное меню.
onloadedmetadata	Скрипт выполняется, когда метаданные (размеры или длительность) загружаются.
onloadeddata	Вызывается при загрузке медиаданных.
onloadstart	Вызывается, когда браузер только начинает загружать медиа данные с сервера.
onpause	Вызывается при воспроизведении медиа данных.
onplaying	Вызывается при воспроизведении медиа данных.
onprogress	Событие onprogress происходит, когда браузер загружает указанное аудио/видео.
onratechange	Вызывается при изменении скорости воспроизведения медиа данных.
onseeked	Вызывается, когда атрибут seeked в теге audio или video изменяет значение с true на false.

onseeking	Вызывается, когда атрибут seeking в тегах audio или video изменяет значение с false на true
onstalled	Скрипт выполняется когда браузер по какой-либо причине не может получить медиаданные.
onsuspend	Скрипт выполняется, когда по какой-либо причине загрузка данных приостановлена до его полной загрузки.
ontimeupdate	Вызывается при изменении позиции воспроизведения элемента <audio> или <video>.
onvolumechange	Вызывается при изменении громкости звука.
onwaiting	Вызывается когда следующий кадр при воспроизведении медиа данных недоступен, но браузер ожидает, что он скоро загрузится.
ontoggle	Вызывается, когда пользователь открывает или закрывает <details>.
onerror	Вызывается, если при загрузке элемента произошла ошибка.

### Урок 3.4. Первые скрипты.

Где упоминалось о команде: «`console.log("Hello World");`», эта команда служит для отображения значения консоли. Вот все команды, которые можно использовать:

```
console.log("Hello World");// Отображение значения в консоли
/*
Где "Hello World" – это значение.
Где; - это значит, что команда окончена. Знак ";"
Обязательно
нужно ставить после того, когда была написана команда.
Где "/" или "/* текст */" – это комментарии.
“//” – комментарий всего на один срок, а "/* текст */" –
комментарий на несколько строк.
*/
```

```
alert("Hello World");// Отображение сообщения в браузере */
window.alert("Hello World");// То же самое, как "alert("Hello
World");//"
return confirm("Hello World");// Отображение сообщения в
браузере с кнопками "Да" или "Нет" */
```

В JavaScript, как и на всех языках программирования, переменные.

Переменные служат для сохранения значений у себя. Сохранять значения



можно в константе, которые никогда не изменяются или в переменной, которая может изменяться. Вот пример создания переменных и констант:

```
const cnst = "Hello World";// Созданная константа
let var = "Hello World";// Созданная переменной
/*
const, var и let – объявления о создании переменных.
"Hello World" – значение переменных.
cnst и var – название переменных
Название переменных может быть любым, но не может
называться служебными словами, такие как: let,
class, return и function. Неанглийские буквы в названиях
переменных разрешены, но не рекомендуются.
Переменные с именами "apple" и "APPLE" – это две разные
переменные, ибо регистр имеет значение. Также переменные
не
могут иметь знаки (такие как: ., ., ?)
*/

// Можно делать следующим образом:
let var2;
var2 = "Hi";

// Таким образом можно и изменять значение переменных
let var3 = "Hello";// Создание новой смынной и
// Задание значения для новой
// переменной
var3 = "Hi";// Изменение значения переменной
```

Все значения переменных этого кода – текстовые. JavaScript имеет следующие типы значений:

Тип значения	Пример
string (текстовые)	"Hi"; "Text"; "Hello World"
number (числовые)	5; 6.24; 100.00; 3.6; 9; 10
boolean (Истина)	<b>true</b> или <b>false</b>
array (Списки)	[3, "Hi", "World", false]

Вот код с переменными разных типов значений:

```
let str = "Hello";
let num = 5.4;
let bool = true;
let arr = ["Hi", ["Hello World",4,true],3,false];
// Продолжение кода на следующей странице
```

/\* array (Список) - тип значения, который может вместить в себя несколько значений (в том числе и переменных, ибо переменная работает как ссылка на значение). Если сделаем список из переменных, содержащихся в этом коде, такие как: str; num; bool; arr, то в коде мы напишем как:

```
[str, num, bool, arr], то JavaScript будет это понимать, как: ["Hello", 5.4, true, ["Hi", ["Hello World", 4, true], 3, false]]
```

Чтобы узнать, какой тип значения имеет переменная, можно сделать следующим образом: \*/

```
let typeOfStr = typeof str; /* typeof помогает узнать тип значения переменной.
```

Мы этого не можем узнать, потому что мы хранили информацию в переменной. Для того чтобы увидеть эту информацию, нужно вывести значение этой переменной в консоли: \*/

```
console.log(typeOfStr)
```

/\* Осталось с нами узнать переменные – это о выборе элемента из списка. Для выбора элемента из списка используем такое понимание как индекс. Индекс – это элемент перечисляемого множества, указывающий на определенный элемент массива. Обычно это неотъемлемое число. В некоторых языках отрицательные индексы используются для подсчета элементов в обратном направлении (начиная с конца массива). Для выбора последнего элемента из списка используем индекс [-1], используем [-2] для пред-последнего, используем [-3] для пред-последнего, и т.д. Это что касается выбора элемента по списку от последнего до первого. Для поиска элементов от начала до конца списка мы вычитаем 1 из числа обычной упорядоченности элемента. Например: первый элемент – [0], второй элемент – [1], третий элемент – [2] и т.д. Вот переменная со списком и переменные, показывающие элементы по индексу: \*/

```
let arrExample = ["First", "Second", "Third", "Fourth"];
```

```
let var1 = arrExample[0]; // Сохраняет значение в переменной, как: "First"
```

```
let var2 = arrExample[2]; // Сохраняет значение в переменной, как: "Second"
```

```
let var3 = arrExample[-1]; // Сохраняет значение в переменной, как: "Fourth"
```

```
let var4 = arrExample[-3]; // Сохраняет значение в переменной, как: "Second"
```

```
arrExample[2] = "3rd"; // Изменяет элемент по индексу [2] на "3rd"
```

```
let var5 = arrExample.push("First"); // Создает элемент "First" // в конце списка
```

```
let var8 = arrExample.unshift("Second"); // Создает элемент "Second" // в начале списка. Продолжение кода на следующей странице.
```

```
let var6 = arrExample.pop();// Сохраняет значение последнего
элемента
// и удаляет последний элемент списка
let var7 = arrExample.shift();// Сохраняет значение первого элемента
// и удаляет первый элемент списка
```

Теперь перейдем к диалоговому сообщению с кнопками OK и Cancel, которое появляется в браузере:

```
let var1 = confirm("Вопрос");// Создание диалогового сообщения
// с кнопками "OK" и "Cancel"
alert(var1);/* Если пользователь нажал на "OK", то
var1=true, если пользователь нажал на "Cancel", то
var1 = false */
```

Теперь разберем использование элементов по id. В общем, классы используют для CSS, а id для JavaScript. Но id можно использовать в CSS и JavaScript, так и классы можно использовать в CSS и JavaScript. Вот как можно получить теги через id или через классы:

```
<!DOCTYPE html>
<html>
  <head><meta charset="UTF-8"></head>
  <body>
    <p id="blinking" class="cls">
Text
    </p>
    <script>
      // Берет теги из id, указанного в функции:
      const p=document.getElementById("blinking")

      // Берет теги из class, указанные в функции:
      const p2=document.getElementsByClassName("cls")

      setInterval(function() {
        if(p.style.fontSize!="10px") {
          p.style.fontSize="10px" // Изменение шрифта
          p.style.color="Red" // Изменение цвета
          p.textContent="hi" // Изменение текста
        }else{
          p.style.fontSize="20px"
          p.style.color="Yellow"
          p.textContent="Hello"
        }
      },XXX)/* setInterval выполняет функцию или команду
постоянно и после завершенной команды, ждет
XXX миллисекунд, а затем дальше делает эту
функцию или команду. Продолжение кода на следующей странице. */
```

```
p.remove();// Удалить элемент из id="blinking"
</script>
</body>
</html>
```

### Урок 3.5. Операторы, условия, разветвления и циклы.

Операторы делятся на базовые и логичные. Логические операторы обычно используются с булевыми (булевыми) значениями; однако значение, которое они возвращают, также логично. Вот таблица с логическими операторами:

Оператор	Проверка
==	Равняется
===	Строго равняется
!=	Не равно
!==	Строго не равняется
>	Меньше
<	Больше
>=	Меньше или равно
<=	Больше или равно
	ИЛИ
&&	ТА
!	НЕТ

Вот какие будут булевы значения у переменных:

```
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
```

Вот таким образом переменные могут сохранять значения, которые могут работать как условия.

Кстати говоря! Как вы и заметили, что есть оператор «Равен» и «Сурово равно». Эти два оператора отличаются тем, что оператор «Равен» может сравнить значение, не обращая внимания на разный тип значений, то оператор «Строго равно» будет сравнивать еще и тип значения. Такая похожая разница между операторами «Не равна» и «Сурово не равна».

Теперь рассмотрим разветвления на JavaScript. Разветвление – схема алгоритмов, при котором, если условие является истиной, то делается по

условию, а в противном случае будет выполняться другое действие. Вот как выглядят разветвления в JavaScript:

```
// Неполное ветвление: ЕСЛИ (...) ТО {...}
if(/* Условие */) {
  // Список команд
}

// Полное ветвление:
// ЕСЛИ (...) ТО {...}, ИНАЧЕ {...}
if(/* Условие */) {
  // Список команд
}else{
  // Список команд
}

// Полное ветвление:
// ЕСЛИ (...) ТО {...}, ИНАЧЕ ЕСЛИ (...) ТО {...}, ..., ИНАЧЕ {...}
if(/* Условие */) {
  // Список команд
}else if{
  // Список команд
}else{
  // Список команд
}
```

Теперь поговорим о базовых операторах. Базовые операторы служат для выполнения математических действий, такие как: умножение; деления; добавить; тому подобное. Вот использование их в коде:

```
// При использовании текстовых
var varText1 = "Hello";
var varText2 = "World";
// var – такой же, как let

var TextSum = varText1 + varText2;
/* В переменном значении будет: "HelloWorld" */

var TextMultiply = varText1 * 3;
/* В переменном значении будет: "HelloHelloHello"
Где было указано 3 – мы указывали число, потому что
текст умножать на текст невозможно. Но
текст умножать на целое положительное число можно.*/

// Продолжение кода на следующей странице
```

```
// Что касается числовых
var varNum1 = 4;
var varNum2 = 3;

var plus = varNum1 + varNum2;// Добавление
/* В переменном значении будет: 7 */

// Продолжение кода на следующей странице
var minus = varNum1 - varNum2;// Вычитание
/* В переменной значение будет: 1 */

var multiply = varNum1 * varNum2;// Умножение
/* В переменной значение будет: 12 */

var divide = varNum1 / varNum2;// Деления
/* В переменной значение будет: 1.3333333333333333 */

var remainder= varNum1 % varNum2;// Получение остатка деления
/* В переменной значение будет: 1 */

var degree= varNum1 ** varNum2;// Возвести в степень
/* В переменной значение будет: 64 */
```

Также у JS есть применение. Пусть у нас будет переменная varApply, которая сохраняет у себя число 5. Вот как будет работать применение:

Название применения	1-ый способ	2-ой способ	Результат
Применить добавление (+=)	<code>varApply += 5;</code>	<code>varApply = varApply + 5;</code> 5+5	10
Применить вычитание (-=)	<code>varApply-= 5;</code>	<code>varApply = varApply + 5;</code> 5-5	0
Применить умножение (*=)	<code>varApply*= 5;</code>	<code>varApply = varApply + 5;</code> 5*5 или 5x5	25
Применить деление (/=)	<code>varApply/= 5;</code>	<code>varApply = varApply + 5;</code> 5/5 или 5:5	1
Применить получение доли деления (%=)	<code>varApply%= 5;</code>	<code>varApply = varApply + 5;</code> 5%5	0
Применить возведение к степени (**=)	<code>varApply**= 5;</code>	<code>varApply = varApply + 5;</code> 5^5	3125

Применить добавление единицы (varApply++)	<code>varApply++;</code>	<code>varApply = varApply + 1;</code>	6
Применить вычитание единицы (varApply--)	<code>varApply--;</code>	<code>varApply = varApply - 1;</code>	4

В JavaScript, как и на всех языках программирования, есть повторяющиеся через условие (Это while) или несколько раз (Это for). Для начала рассмотрим цикл for. Вот код с этим циклом:

```
for(Initialization;Condition;Increment/Decrement) {
  // Список кодов
}
/*
Initialization: Здесь вы инициализируете новый счетчик, который
используется в цикле for. Выполняется только один раз.
Condition: Выражение, которое проверяется перед каждой итерацией,
подобно циклу while.
Increment/Decrement: Операции, выполняемые над счетчиком
в конце каждой итерации цикла.

Пример написания той команды:
for (let i = 1; i < 5; i++) {
  console.log("Loop iteration:", i);
};

let i = 1: Инициализация, где мы создаем переменную i внутри цикла
for. Эта операция выполняется один раз.
i < 5: Условие, проверяемое перед каждой итерацией.
i++: Выражение для увеличения выполняется после каждой итерации.
console.log("Loop iteration:", i);: Тело цикла for.
*/
```

Теперь о цикле while. Вот код:

```
while(/* Условие */) {
  // Список кодов
}

// Можно и таким образом писать:
do{
  // Список кодов
}while(/* Условие */);
```

Теперь перейдем к конвертации значений. Например: у нас есть переменная, имеющая текстовый тип значения, то нам нужно получить его числовой тип значения. Вот такой код:

```

var textNum = "5";// Текстовый тип значения
var numNum = 5;// Числовой тип значения
var numNum2 = parseInt(textNum);/* Преобразование
текста к целому числу. Продолжение кода на следующей странице */
var numNum3 = parseFloat(textNum);/* Преобразование текста
до десятичного числа */
var textNum2 = numNum.();/* Преобразование числовое
к текстовому значению */

```

### Урок 3.6. функции.

Пример, как можно создавать функции на JS:

```

function name(a,b,c, ...) {
  // Список команд
};
// Где name – это название функции
// Где a, b, c, ... – это переменные, которые может
принимать функция

```

Теперь сделаем функцию, принимающую значение и показывающую результат в консоли:

```

function name(a,b,c) {
  console.log(a + b * c);
};
name(5,10,4)// В консоли будет: 45, ибо 5+10*4=45

```

Но это показать результат в консоли, потому что можно сделать так, чтобы функция возвращала результат:

```

var varFun1;
function name(a,b,c) {
  return a + b * c;// Возвращает значение функции
};
varFun1 = name(6,30,2)
// В переменной varFun1 будет: 66, ибо 6+30*2=66
console.log((45,2,5))// В консоли будет: 55

```

А это HTML-код, в котором есть поле ввода и кнопка для отображения диалогового сообщения, имеющего текст с поля ввода:

```

<!DOCTYPE html>
<html><head>
  <meta charset="UTF-8">
  <title>JavaScript</title>
</head>
<body>
  <input id="input">
// Продолжение кода на следующей странице

```



```
<button onclick="a()">Alert</button>
<script>
function a() {
const textFromInput=document.getElementById("input")
let text=textFromInput.value;
// textFromInput.value берет значение из <input>
alert(text);};
</script>
</body>
</html>
```

## Урок 3.7. Рандом.

### Основы генерации случайных чисел

В JavaScript для создания случайных чисел используется встроенный объект Math с методом random(). Метод Math.random() возвращает случайное число в диапазоне от 0 (включительно) до 1 (не включительно).

```
let randomNumber = Math.random();
console.log(randomNumber);/* Случайное число между 0
(включительно) и 1 (не включительно) */
```

### Генерация случайных целых чисел

Чтобы сгенерировать целое случайное число в определенном диапазоне, нужно использовать Math.random() вместе с математическими операциями.

Пример: Случайное число от min до max включительно:

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

let randomInt = getRandomInt(1,10);
console.log(randomInt);// Случайное число между 1 и 10
включительно
```

- Math.random() генерирует число от 0 до <1.
- Math.random()\*(max - min + 1) расширяет диапазон до [0, max - min + 1).
- Math.floor() округляет число до ближайшего меньшего целого.
- Добавление min сдвигает диапазон к [min, max].

### Генерация случайных чисел с плавающей запятой

Чтобы получить случайное число с плавающей запятой в конкретном диапазоне, можно использовать Math.random() без округления.

Пример: Случайное число от min до max

```
function getRandomFloat(min,max) {  
  return Math.random()*(max - min)+ min;  
}  
  
let randomFloat = getRandomFloat(1.5,5.5);  
console.log(randomFloat);/* Случайное число между 1.5  
(включительно) и 5.5 (не включительно) */
```

### Генерация случайных элементов из массива

Для выбора случайного элемента из массива используют Math.random() вместе с Math.floor().

Пример: Случайный элемент из массива:

```
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);// Случайный фрукт из массива
```

### Быстрые советы

Не используйте Math.random() для криптографически безопасных случайных чисел. В таких случаях используйте crypto.getRandomValues().

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

Избегайте использования Math.random() для важных функций безопасности, таких как генерация паролей или токенов. Лучше использовать специализированные библиотеки.

### Итог

В JavaScript генерация случайных чисел достаточно проста благодаря методу Math.random(). Для более сложных случаев, таких как случайный выбор из массива или генерация чисел с плавающей запятой, используйте комбинирование математических функций. Для криптографических нужд используйте API crypto.

## Урок 3.8. Скрипты для веб-программирования и фреймворки.

Существует несколько популярных фреймворков JavaScript, которые широко используются для создания веб-приложений. Вот некоторые из них:

### React

**Разработчик:** Facebook

React – это библиотека JavaScript для построения пользовательских интерфейсов. Она позволяет создавать компоненты, которые могут повторно использоваться, и управлять состоянием приложения.

- **Особенности:**
  - Использует JSX для описания компонентов UI.
  - Односторонняя привязка данных.
  - Виртуальный DOM для улучшения производительности.
  - Поддержка расширений, таких как React Router для маршрутизации.

Его пример:

```
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

**Разработчик:** Google

Angular – это платформа для построения веб-приложений, которая включает в себя фреймворк, библиотеки и инструменты.

- **Особенности:**

- Использует TypeScript в качестве основного языка.
- Двухсторонняя привязка данных.
- Модульная архитектура.
- Встроенные сервисы для запросов HTTP, маршрутизации и других задач.

Его пример:

```
//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

**Разработчик:** Evan You

Vue.js – это прогрессивный фреймворк для построения пользовательских интерфейсов, который легко интегрируется в существующие проекты.

- **Особенности:**

- Легкая интеграция с другими библиотеками.
- Односторонняя и двухсторонняя привязка данных.
- Виртуальный DOM.
- Поддержка компонентов и директив.

Его пример:

```
<!-- 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

**Разработчик:** Rich Harris

Svelte – это инструмент для построения быстрых веб-приложений, отличающийся тем, что выполняет большую часть работы во время компиляции, а не в браузере.

- **Особенности:**
  - Без виртуального DOM.
  - Более простой синтаксис.
  - Высокая производительность.
  - Низкий размер пакетов.

Его пример:

```
<!-- 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

**Разработчик:** Ember Core Team

Ember.js – это фреймворк для построения амбициозных веб-приложений с четкой структурой и конвенциями.

- **Особенности:**

- Мощная система маршрутизации.
- Двухсторонняя привязка данных.
- Большое количество встроенных инструментов и вспомогательных библиотек.
- Строгой организации кода.

Его пример:

```
<!-- 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);
```

Каждый из этих фреймворков имеет свои преимущества и особенности и выбор зависит от конкретных потребностей вашего проекта.

Есть также код с выставлением значения ползунка. Пример кода с выставлением значения ползунка находится на следующей странице.

```
<!-- Выставляет значение ползунка -->
<!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;//Выставляет значение
    // ползунка на 70%
  </script>
</body>
</html>
```



## IV РАЗДЕЛ. ИСПОЛЬЗОВАНИЕ БАЗЫ ДАННЫХ.

### Урок 4.1. Что такое база данных.

База данных (БД) – это организованный набор данных, хранящихся и управляемых электронными средствами. Они позволяют хранить, искать, изменять и извлекать информацию. Основные компоненты базы данных включают в себя:

1. **Данные:** Информация, хранящаяся в базе данных.
2. **Система управления базами данных (СУБД):** Программное обеспечение, используемое для управления базой данных, например MySQL, PostgreSQL, Oracle, SQL Server.
3. **Таблицы:** Основные структуры в реляционных базах данных, где данные хранятся в строках и столбцах.
4. **Запросы:** Инструменты для извлечения данных из базы данных часто используются языки запросов, такие как SQL.
5. **Схема:** Структура, определяющая организацию данных в базе, включая таблицы, поля и взаимосвязи между ними.

Базы данных используются для хранения разнообразной информации, от личных данных до коммерческих транзакций и научных исследований. Они являются основой для большинства информационных систем и приложений, требующих обработки больших объемов данных.

### Урок 4.2. Реляционные и нереляционные базы данных.

Реляционные базы данных – база данных, которая имеет табличный вид, а нереляционные базы данных – база данных, которая имеет текстовый вид.

Реляционные базы данных		Нереляционные базы данных								
<table><tr><th>First Name</th><th>Last Name</th></tr><tr><td>Joe</td><td>Биден</td></tr><tr><td>Emmanuel</td><td>Macron</td></tr><tr><td>Justin</td><td>Trudeau</td></tr></table>		First Name	Last Name	Joe	Биден	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	Биден									
Emmanuel	Macron									
Justin	Trudeau									

## Урок 4.3. Использование JSON с помощью JavaScript.

Пусть у нас таблица JSON будет находиться в странице, которая имеет JS-скрипт. Вот код страницы:

```
<!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>
    // Встроенные JSON данные
    const data= {
      "users":[
{"name": "John Doe", "age": 30},
{"name": "Jane Doe", "age": 25}
]
    };

    // Функция для отображения данных на странице
    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);
      });
    }

    // Отобразить данные после загрузки страницы
    window.onload={()=> displayData(data);
  </script></body></html>
```

## Урок 4.4. Что такое SQL.

SQL (Structured Query Language) – это язык программирования, используемый для управления и манипулирования реляционными базами данных. SQL позволяет выполнять следующие действия:

1. **Запросы данных:** Получение данных из одной или нескольких таблиц базы данных.

```
SELECT * FROM table_name;
```

2. **Вставка данных:** Добавление новых записей в таблицу.

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

3. **Обновление данных:** Изменение существующих записей в таблице

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

4. **Удаление данных:** Удаление записей из таблицы.

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

5. **Создание и модификация структуры базы данных:** Создание новых таблиц, изменение структуры существующих таблиц.

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

6. **Управление доступом к данным:** Установка прав доступа для пользователей.

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

SQL является стандартом для работы с реляционными базами данных, такими как MySQL, PostgreSQL, SQLite, Microsoft SQL Server и другие.

## Урок 4.5. Получение данных, сортировка, условия и операторы.

Мы будем работать с базой данных country, содержащей таблицу country. Давайте посмотрим на эту таблицу:

id	name	continent	region	surfacearea	capital	population
1	Japan	Asia	Eastern Asia	377829	Токио	126714000
2	Латвия	Европа	NULL	64589	Рига	2424200
3	Мексика	North America	Central America	1958201	Мексика	98881000
4	Нидерланды	Европа	Western Europe	41526	Амстердам	15864000
5	Португалия	Европа	Southern Europe	91982	Lisbon	9997600
6	Швеция	Европа	Nordic Countries	449964	Stockholm	NULL
7	Украина	Европа	Eastern Europe	603700	Kyiv	50456000
8	United States	North America	NULL	9363520	Washington	278357000
9	Тайланд	Asia	NULL	513115	Bangkok	61399000
10	Paraguay	South America	North America	406752	Asunción	5496000
11	Philippines	Asia	Southeast Asia	300000	Manila	75967000
12	Новый Зеланд	Oceania	Australia and New Zealand	270534	Wellington	18886000
13	Norway	Европа	Nordic Countries	323877	Oslo	4478500
14	Монако	Европа	Western Europe	1.50	Монако	NULL
15	Malta	Европа	Southern Europe	316	Valletta	380200

SELECT – это оператор, возвращающий набор данных (выборку) из базы данных. Для того чтобы использовать оператор SELECT, мы должны указать, что именно мы хотим выбрать и откуда мы хотим выбрать.

```
SELECT континент FROM country;
```

В предыдущем запросе мы использовали выражение SELECT, чтобы получить единственный столбец continent из таблицы country. Нам нужно указать имя столбца сразу после ключевого слова SELECT, а после

ключевого слова FROM указываем имя таблицы, из которой мы хотим получить данные.

Давайте узнаем немного больше об этой базе данных. База данных country содержит одну таблицу под названием country. Давайте посмотрим на эту таблицу.

Эта таблица содержит 15 строк. Другими словами, у нас есть 15 различных записей для разных стран.

А как насчет столбцов? Здесь у нас есть 7 столбцов, таких как id, name, континент, регион, SurfaceArea, capital и population.

1. id – номер записи в этой таблице;
2. name – название страны;
3. continent – название континента, на котором находится страна;
4. region – название региона страны;
5. SurfaceArea – площадь поверхности страны;
6. capital – столица страны;
7. population – население страны.

Мы можем получить несколько столбцов с помощью оператора SELECT. Единственное отличие состоит в том, что после слова SELECT нам нужно будет указать несколько имен столбцов, которые должны быть разделены запятой. Рассмотрим пример, где мы получаем три столбца из таблицы стран:

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

id	name	capital
1	Japan	Токио
2	Латвия	Рига
3	Мексика	Мексика
4	Нидерланды	Амстердам
...	...	...
15	Malta	

Обратите внимание, что запрос SELECT возвращает все строки в заданном столбце. Однако что делать, если нам не нужны все значения из столбца (поскольку они могут дублироваться), а нужны только уникальные значения? Для таких случаев удобно использовать ключевое слово DISTINCT. Причем это ключевое слово используется непосредственно перед названиями столбцов. Рассмотрим пример:

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

Результат находится на следующей странице.

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

Мы знаем, что оператор **SELECT** выбирает все строчки из таблицы всех или определенных столбцов. Однако что делать, если нам нужно получить только определенное количество строк, например?

**Заметьте.** В разных системах управления это реализовано с разным синтаксисом. Поскольку мы используем PostgreSQL, мы должны использовать оператор **LIMIT**.

В следующем примере мы вытаскиваем первые 7 строк (в нашем случае - столицы стран) из столбика:

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

<b>capital</b>
Токио
Рига
Мексика
Амстердам
Lisbon
Stockholm
Kyiv

Команда для получения всех данных из таблицы:

```
SELECT * FROM country;
```

Давайте поймем, что означает "группировка данных", на простом примере таблицы работников:

department	employee_id	first_name	hire_date	last_name	salary
Engineering	1	Джон	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	

Теперь давайте представим, что у нас есть задача "определить количество работников в каждом департаменте." Для этого мы сгруппируем данные по столбцу department и используем агрегацию с помощью функции COUNT(\*).

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

Итак, как вы можете увидеть, синтаксис для группировки данных выглядит следующим образом:

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

Примечание: AGG\_FUNC означает агрегатные функции типа MAX, MIN, COUNT и т.д.

Этот синтаксис существует для поиска определенных значений с помощью агрегатных функций в специфических колонках.

Рассмотрим другой пример: наша задача состоит в нахождении отдела с самой высокой средней зарплатой.

Чтобы получить такие данные, нам нужно группировать данные по колонке `department` и затем использовать функцию `AVG()`, чтобы рассчитать среднюю зарплату:

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

Обратите внимание, что мы не будем использовать эту таблицу в задачах; таблица `employees` будет использоваться исключительно для демонстрации примеров синтаксиса и их применения.

В этом курсе мы будем работать с базой данных системы монреальского метро, содержащей таблицу `metro_travel_time`.

Эта таблица будет содержать информацию о линии станции (`line_name`), ее названии (`station_name`), и количестве времени, которое требуется поезда для путешествия от одной станции до следующей (`time_to_next_station`).

Вот как выглядит эта таблица и предварительный просмотр данных в ней:

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

Как вы видите, это не сложная таблица. Давайте подумаем, где мы можем применить группировку здесь.

Самый очевидный вариант – это группировка по цвету линий метро. Это означает, что мы можем агрегировать данные, группируя их по цвету метро.

Теперь давайте попрактикуемся в группировке, выполняя задания.

Для статистического анализа нам было поручено сосчитать количество станций на каждой линии и упорядочить их по возрастанию количества станций на каждой из линий метро.

Для этого нам нужно найти количество станций на каждой из линий метро, а затем отсортировать их от наименьшего количества станций до самого большого.



Таким образом, строительная компания поймет, каким линиям метро следует уделять первостепенное внимание для добавления станций.

Для нас важно понять порядок записи клюз, в частности, где должен быть размещен GROUP BY клюз.

Итак, порядок выглядит так:

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

Из этого порядка очевидно, что оператор GROUP BY должен быть написан ПОСЛЕ оператора WHERE (или после FROM таблицы, если в вашем запросе не используется фильтрация с использованием SELECT) и также К оператору ORDER BY.

Давайте рассмотрим пример такого порядка инструкций, используя нашу таблицу employee. Предположим, нам нужно получить количество работников в каждом department, чья salary выше 70000, и отсортировать их по росту:

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

Примечание: Следует отметить, что класс LIMIT всегда пишется в конце. Так вы легко запомните его местоположение в запросе.

Строительная компания решила увеличить количество станций на Желтой линии метро.

Наша следующая задача – найти время оборота для каждой линии. Для компании важно иметь возможность закрыть Желтую линию для технического обслуживания и расширения путем добавления новых станций метро, так что крайне важно не создавать чрезмерные неудобства для пассажиров.

Поэтому нам нужно определить общее время оборота поезда, подытожив время к каждой станции (используя функцию SUM()).

**Примечание:** Если мы просто подсчитаем сумму времени на каждую станцию, это будет время движения поезда от одной конечной станции к другой. Однако также важно знать общее время оборота поезда по всей линии метро. Чтобы добиться этого, мы должны умножить сумму на 2.

Чтобы понять, как выполнить эту задачу, рассмотрим пример с таблицей `employees`.

Допустим, нам нужно найти отдел с самой высокой среднемесячной зарплатой.

Для этого можем использовать следующий запрос:

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

Итак, мы получаем необходимые данные в результате.

Вот случай: школа рекомендовала вас, и школа заинтересовалась вами, поскольку у них также есть несколько задач для вас. Но сначала давайте ознакомимся с таблицей, которую они предоставили:

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

Как вы можете увидеть, в школе учиться в общей сложности 100 учеников, информация о которых представлена в данной таблице. В столбце `class_letter` есть информация, которая может иметь 3 варианта: A, B или C. Также здесь указано название предмета (`subject_name`) и оценка ученика (`grade`). Таблица проста и содержит оценки по экзаменам по разным предметам.

Посмотрим, сколько учащихся находится в каждом классе с помощью следующего запроса:

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

Школа удовлетворена нашей работой и соглашается продолжить сотрудничество.

Теперь у них есть для нас новая задача. Топ-10 учеников с высоким средним баллом получают в качестве вознаграждения поездку в научный центр. Одним из обязательных условий является получение оценки выше 90 по математике на экзамене. Чтобы найти таких студентов, они обратились к вам.

Посмотрим, что нам нужно сделать, используя нашу таблицу `employees` в качестве примера.

Предположим, нам нужно узнать, в каких департаментах есть работники, принятые на работу до 2019 года и средняя заработная плата в этих департаментах. Для выполнения такой задачи можно использовать следующий запрос:

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

Как вы можете увидеть таких работников всего 3, и мы использовали необходимые инструменты для достижения этого результата. Ваша задача будет очень похожа, я уверен, что вы с этим справитесь!

Вот предварительный просмотр таблицы `student_grades`, с которой мы работаем:

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

Школа очень благодарна за вашу работу, и теперь у нас есть новая задача.

Оказалось, что некоторые студенты сдавали дополнительные экзамены, когда должны были сдать только один. Школа подозревает его в нечестности, поскольку каждый студент должен иметь только одну оценку.

Нам поручили выяснить фамилии этих студентов и передать их администрации школы, чтобы они смогли принять необходимые меры.

Давайте вместе подумаем, как мы можем это сделать. Вы можете начать с того, что мы можем сделать это с помощью клавиши WHERE, и это будет выглядеть примерно так:

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

Но, как вы можете увидеть, мы получаем ошибку, что указывает на то, что мы не можем использовать агрегатные функции внутри клязы WHERE. Вот где нам понадобится кляза HAVING.

Давайте поймем, что это такое и как использовать, на примере из нашей таблицы employeee.

Предположим, нам нужно получить отделы, где средняя заработная плата сотрудников ниже \$70,000 в год.

Для этого нам понадобится использовать агрегатную функцию и клязу HAVING.

Давайте посмотрим, как мы можем это сделать:

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

Мы получили один ответный отдел, используя класс HAVING, где мы установили условие для колонки, по которой группировали данные.

Примечание: Чтобы использовать агрегацию данных в классе HAVING, необходимо иметь группировку данных в нашем запросе. Как в запросе выше, мы группировали данные по колонке department.

Давайте рассмотрим более обобщенный синтаксис класса HAVING и когда его лучше использовать:

```
SELECT column1, column2... --(по желанию)
FROM table
GROUP BY column1
HAVING AGG(column_n) --(условие);
```

Давайте также кратко рассмотрим главное отличие между клозами WHERE и HAVING и когда следует использовать каждый из них:

Клоз WHERE используется для агрегации данных, тогда как клоз HAVING применяется после агрегации данных;

Клоз WHERE пишется в GROUP BY, в то время как клоз HAVING записывается после GROUP BY.

Ранее школа проводила соревнования для учащихся, составлявших Математику, и некоторые из них получили награды. Теперь школа хочет убедиться, что не будет учеников, которые обманывают и сдают более одного экзамена, включая экзамен по математике.

Поэтому школа попросила нас определить фамилии тех учеников, которые сдали более одного экзамена, одним из которых был Mathematics.

Вот запрос из нашей предыдущей задачи, который вы можете использовать в качестве примера:

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

Где мы указывали знак «>» – это знак неравенства. Вот как можно задавать знаки неравенства в SQL при условии:

Знак	Что значит
=	Равняется
<>	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Раньше мы работали на разные компании и выполняли запросы SELECT для их нужд. Однако нам необходимо научиться создавать и модифицировать таблицы.

Давайте перейдем прямо к делу!

Таблицы создаются с помощью оператора CREATE, который имеет схожую структуру оператору SELECT, но вместо выборки данных, он создает данные.

Посмотрим на синтаксис:

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

Примечание: когда вы запускаете эти примеры, вы не получите ни одного вывода, поскольку эти примеры создают только новую таблицу. Если вы попытаетесь запустить код снова, вы получите сообщение об ошибке, которое будет говорить, что таблица уже существует. Эти фрагменты кода являются примерами, и позже, в задании, в эти таблицы будут вставлены данные и отображены на экране, чтобы вы могли увидеть, что все работает.

Теперь разберемся, что написано выше.

Этот запрос создаст пустую таблицу с двумя колонками: id и some\_info.

Обратите внимание на используемые типы данных. Слова INT или VARCHAR обозначают тип для каждой колонки. К примеру, INT представляет целочисленные данные, тогда как VARCHAR(50) представляет текст с максимумом 50 символов.

Сейчас мы не будем углубляться во все типы данных, поскольку их достаточно много. Мы сосредоточимся на основных типах данных в этом разделе и рассмотрим каждый из них по мере того, как будем продвигаться в обучении!

Например, создадим другую таблицу с разными типами данных:

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

С помощью этого запроса мы создаем пустую таблицу, содержащую информацию о пользователях, включая:

1. ID с типом данных целое число;
2. Информацию об имени с типом данных VARCHAR(50);
3. Информация о дате рождения, с типом данных DATE.
4. Информацию о зарплате, с типом данных число с плавающей запятой;
5. Является ли пользователь активным, с типом данных, принимающим только значение true или false.

Ограничение:

Вы могли заметить, что рядом с каждым значением ID мы размещаем слова PRIMARY KEY. Это называется ограничением и означает ограничение, наложенное на эту колонку.

Например, PRIMARY KEY обеспечивает уникальность и идентифицирует каждую строку таблицы. В таблице может быть только одна такая колонка.

Также существуют другие ограничения, например:

- NOT NULL: Гарантирует, что колонка не будет содержать значения NULL;
- UNIQUE: обеспечивает уникальность всех значений в колонке или комбинации колонок;
- DEFAULT: Устанавливает значение по умолчанию для колонки, если при вставке данных для этой колонки не указано значение.

Это не все ограничения, которые используются, но пока нам нужны именно эти.

Рассмотрим пример, где мы модифицируем предыдущую таблицу:

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

Теперь столбец name не может иметь пустых или null значений, а столбец salary по умолчанию имеет значение 50000.

Таким образом, можно использовать ограничения для контроля столбцов таблицы во время ее создания.

В предыдущей главе мы научились создавать таблицы.

Но представим ситуацию, когда нам нужно добавить столбец в существующую таблицу. Было бы довольно глупо удалять таблицу (особенно если она уже содержит какие-то данные) и затем создавать новую, снова заполняя ее данными.

Поэтому в этой главе мы рассмотрим операцию ALTER.

Давайте посмотрим, как использовать эту операцию:

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

Как вы можете увидеть это скрипт для создания таблицы из предыдущего раздела.

Далее следуют две операции ALTER. Первая операция добавляет в таблицу столбец price, устанавливая значение по умолчанию 300 для этого столбца. Вторая операция удаляет этот столбец.

Синтаксис очень прост:

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

Синтаксис действительно довольно прост.

Примечание! С помощью оператора ALTER можно выполнять различные операции на уровне схемы в таблице, например добавление или удаление



ограничений, переименование, изменение типов данных и добавление или удаление индексов.

Перейдем к другой операции, а именно операции вставки. Для вставки данных в SQL можно использовать оператор INSERT. Для использования INSERT мы должны указать, в какие колонки мы хотим добавить значение.

Вот как выглядит синтаксис этого оператора:

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

Вы, вероятно, правильно заметили, что это отрывок скрипта из предыдущего раздела, где данные вставляются в таблицу library.

Давайте разберем, что здесь происходит:

1. Сначала пишутся ключевые слова INSERT INTO, за которыми следует table\_name, где будут вставлены данные;
2. Затем открываются скобки и указываются названия столбцов, куда будут вставлены данные; в нашем случае есть 4 столбца;
3. После этого пишется ключевое слово VALUES, и открываются скобки, на которые будут записаны данные;
4. Данные следует записывать в том же порядке, что и названия столбцов, при этом следует соблюдать типы данных. К примеру, вы не можете вставить целочисленное значение в столбец с типом данных VARCHAR;
5. **Скобки закрываются**, а затем ставится запятая, таким образом заполняется одна строка. Вы можете заполнить столько строк, сколько считаете нужным, используя этот метод.

Подытоживая, общий синтаксис оператора INSERT выглядит так:

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

Не забудьте о точке с запятой в конце!

Следует упомянуть еще две операции в DDL: DROP и TRUNCATE.

Давайте кратко рассмотрим, что делает каждая из этих операций:

**DROP:** Используется для удаления объектов базы данных, таких как таблицы, базы данных и индексы.

**TRUNCATE:** Удаляет все строки из таблицы, но сохраняет ее структуру.

Я использовал эти операции для очистки или удаления таблиц, чтобы проверить задачи в предыдущих разделах. Их синтаксис достаточно прост; давайте на них посмотрим:

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

Этот код удалит таблицу `employees` из базы данных. У многих СУБД эта операция требует определенных разрешений, и если вы работаете над проектом, у вас может не быть доступа к такой операции. Вы узнаете о ролях и как их настроить в следующем курсе, который охватывает расширенные концепции SQL.

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

Этот код удалит все строки из таблицы `employees`, полностью очистив ее и сделав пустым. Эта операция не повлияет на структуру таблицы, то есть не изменит колонок или ограничений. Вам также требуются разрешения в СУБД для такой операции, поскольку не каждому должны быть предоставлены способности очищать таблицу.

Используйте эти операции осторожно, так как если у вас нет снимков базы данных, вы не сможете откатить удаление таблицы или обрезку строк.

**Примечание!** Часто разработчики используют мягкое удаление, добавляя новую колонку `is_deleted` с типом данных `BOOLEAN`, и когда некоторые строки удаляются, статус устанавливается как `true` (или `1`). Таким образом, вы можете видеть удаленные данные и не беспокоиться об их целостности.

Вы уже знаете, как очистить таблицу, добавить столбец, вставить данные. Однако для правильного взаимодействия с базой данных необходимо понимать, как обновлять и удалять отдельные строки.

Для этого существует два утверждения и типы запросов: запросы `UPDATE` и `DELETE`.

**UPDATE:** Используется для изменения существующих данных в таблице. Посредством такого запроса мы можем изменить данные в таблице, не влияя на другие строки. Давайте рассмотрим пример с таблицей `medications`, которая находится на следующей странице.

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

Представим, что нам нужно обновить цену на определенный тип лекарства. К примеру, сейчас на Ibuprofen действует 50% скидка, и мы должны изменить цену на этот продукт.

Наш запрос на обновление будет выглядеть так:

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

SELECT *
FROM medications
ORDER BY id;
```

Вот мы обновили таблицу medications, чтобы цена продукта с ID 2 (Ibuprofen) была установлена на 4. После этого мы выбрали все столбцы из таблицы, чтобы убедиться, что столбец цена успешно обновлена. Вы можете заменить любое значение и увидеть, как работает операция обновления в SQL.

В итоге общий синтаксис выглядит так:

```
UPDATE table_name
SET column_name = значение
WHERE какое-то условие;
```

Операция команды DELETE почти схожа по принципу. Однако здесь мы не используем ключевое слово SET, ведь мы ничего не меняем; мы просто удаляем строки. Синтаксис для удаления будет выглядеть так:

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

Но я напому вам, что удаление строк следует производить осторожно, ведь вы не сможете легко их восстановить.

Примечание! Если вы не включите условия WHERE, данные будут обновлены или удалены для всех строк.

