

МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И  
МАССОВЫХ КОММУНИКАЦИЙ РФ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра программной инженерии

**А. Э. БАЖЕНОВ**

# **ОСНОВЫ ВЕБ-РАЗРАБОТКИ**

УЧЕБНОЕ ПОСОБИЕ

Самара, 2023

УДК 000.000

ББК 00.000

Б 00

Рекомендовано к изданию методическим советом ПГУТИ Протокол  
No \_\_ от \_\_. \_\_.20\_\_

Рецензенты:

Матвеева, Е. А.

М 00 Управление информационным ресурсами: учебное пособие /А. Э.  
Баженов. – Самара: ПГУТИ, 20\_\_. – \_\_ с.

Учебное пособие (название) содержит (краткое описание). Разработано  
в соответствии с ФГОС ВО по направлению подготовки/специальности (код и  
наименование) и предназначено для студентов (уровень обучения) (форма  
обучения) для подготовки к (вид занятий: практические, лабораторные,  
самостоятельная работа, зачеты, экзамены и др.)

© Баженов А. Э., 2023 © Поволжский государственный университет  
телекоммуникаций и информатики, 2023

# Содержание

<b>1 HTML.....</b>	<b>8</b>
<b>1.1 Введение в HTML .....</b>	<b>8</b>
1.1.1 Что такое HTML? .....	8
1.1.2 Зачем нужен HTML.....	8
1.1.3 Краткая история HTML .....	8
<b>1.2 Основы HTML.....</b>	<b>8</b>
1.2.1 Структура HTML-документа .....	8
1.2.2 Теги, элементы и атрибуты.....	10
1.2.3 Заголовки .....	10
1.2.4 Абзацы.....	11
1.2.5 Нумерованные списки .....	11
1.2.6 Маркированные списки .....	12
1.2.7 Списки определений .....	12
1.2.8 Гиперссылки .....	13
1.2.9 Изображения.....	13
1.2.10 Комментарии в HTML .....	14
1.2.11 Специальные символы и символы перевода строки.....	15
1.2.12 Таблицы.....	15
<b>1.3 Формы и элементы управления.....</b>	<b>16</b>
1.3.1 Формы .....	16
1.3.2 Текстовые поля, кнопки и переключатели .....	17
1.3.3 Списки выбора и множественный выбор .....	17
1.3.4 Текстовые области и элементы для загрузки файлов .....	17
1.3.5 Атрибуты форм и методы отправки данных.....	17
1.3.6 Валидация и атрибуты HTML5 для форм.....	17
<b>1.4 Семантика и структура .....</b>	<b>17</b>
1.4.1 Семантические теги HTML5.....	17
1.4.2 <header>, <nav>, <main>, <article>, <section> .....	17
1.4.3 <aside>, <footer>, <figure>, <figcaption> .....	17
1.4.4 Заголовки и навигация.....	17
1.4.5 Аудио и видео .....	17
1.4.6 Теги <audio> и <video> .....	17
1.4.7 Атрибуты и поддерживаемые форматы .....	17
1.4.8 Ссылки и закладки .....	17
<b>1.5 Задание для самоподготовки .....</b>	<b>17</b>
1.5.1 Цель .....	17
1.5.2 Описание .....	17
1.5.3 Контрольные вопросы .....	18

<b>2 CSS</b>	<b>19</b>
<b>2.1 Введение в CSS</b>	<b>19</b>
2.1.1 Что такое CSS и зачем оно нужно	19
2.1.2 Основные принципы работы CSS	19
2.1.3 Внедрение CSS в веб-страницу	19
<b>2.2 Синтаксис CSS</b>	<b>19</b>
2.2.1 Селекторы	19
2.2.2 Свойства	19
2.2.3 Значения	19
2.2.4 Комментарии	19
<b>2.3 Внедрение CSS в HTML</b>	<b>19</b>
2.3.1 Внутренние (встроенные) стили	19
2.3.2 Внешние таблицы стилей (external CSS)	19
2.3.3 Встроенные стили в HTML-элементах	19
<b>2.4 Селекторы CSS</b>	<b>19</b>
2.4.1 Селекторы элементов	19
2.4.2 Селекторы классов	19
2.4.3 Селекторы идентификаторов	19
2.4.4 Комбинированные селекторы	19
<b>2.5 Основные свойства и значения</b>	<b>19</b>
2.5.1 Цвет и фон	19
2.5.2 Текст и шрифты	19
2.5.3 Отступы и рамки	19
2.5.4 Позиционирование и размеры	19
<b>2.6 Каскадность и наследование</b>	<b>20</b>
2.6.1 Приоритет селекторов	20
2.6.2 Каскадирование стилей	20
2.6.3 Наследование свойств	20
<b>2.7 Практические примеры</b>	<b>20</b>
2.7.1 Создание навигационного меню	20
2.7.2 Оформление таблиц	20
2.7.3 Создание адаптивного дизайна	20
2.7.4 Работа с анимациями и переходами	20
<b>2.8 Оптимизация и отладка</b>	<b>20</b>
2.8.1 Инструменты для отладки CSS	20
2.8.2 Минимизация и оптимизация кода	20
2.8.3 Работа с браузерными различиями	20
<b>2.9 Расширенные темы</b>	<b>20</b>

2.9.1 Flexbox и Grid Layout.....	20
2.9.2 CSS-препроцессоры.....	20
2.9.3 CSS-фреймворки.....	20
<b>2.10 Задание для самоподготовки .....</b>	<b>20</b>
2.10.1 Цель .....	20
2.10.2 Описание.....	20
2.10.3 Варианты.....	21
2.10.4 Контрольные вопросы .....	22
<b>3 Основы JavaScript.....</b>	<b>24</b>
<b>3.1 Введение.....</b>	<b>24</b>
3.1.1 Что такое JavaScript? .....	24
3.1.2 История развития JavaScript.....	24
3.1.3 Важность изучения JavaScript.....	24
<b>3.2 Основы JavaScript.....</b>	<b>24</b>
3.2.1 Синтаксис и правила написания кода.....	24
3.2.2 Комментарии.....	25
3.2.3 Заключение инструкций в точку с запятой .....	25
3.2.4 Переменные .....	26
3.2.5 Регистрозависимость .....	26
3.2.6 Идентификаторы .....	27
3.2.7 Типы данных.....	27
3.2.8 Типизация .....	28
3.2.9 Операторы и выражения.....	29
3.2.10 Условные операторы и циклы .....	30
3.2.11 Блоки кода .....	31
3.2.12 Функции и замыкания.....	31
<b>3.3 Работа с ошибками и отладка.....</b>	<b>32</b>
3.3.1 Обработка ошибок и исключений .....	32
3.3.2 Отладка с помощью инструментов разработчика .....	33
3.3.3 Типичные ошибки .....	34
<b>3.4 Работа с DOM и HTML .....</b>	<b>34</b>
3.4.1 Основы работы с DOM .....	34
3.4.2 Создание и манипулирование элементами HTML.....	35
3.4.3 Манипуляция стилями и классами .....	35
3.4.4 События.....	35
3.4.5 Обработка событий в JavaScript.....	36
3.4.6 Объект события (Event Object).....	37
<b>3.5 Объектная модель браузера (BOM).....</b>	<b>37</b>

3.5.1 Основные понятия BOM .....	37
3.5.2 Использование BOM для взаимодействия с браузером.....	37
<b>3.6 Взаимодействие с сервером и AJAX.....</b>	<b>38</b>
3.6.1 Основы AJAX .....	38
3.6.2 XMLHttpRequest и его методы.....	38
3.6.3 Fetch API.....	39
3.6.4 JSON: передача и обработка данных.....	39
3.6.5 Плюсы и минусы использования AJAX.....	40
<b>3.7 Работа с формами и валидация данных .....</b>	<b>41</b>
3.7.1 Взаимодействие с разными типами форм.....	41
3.7.2 Валидация входных данных .....	41
3.7.3 Отправка данных на сервер.....	41
<b>3.8 Websocket.....</b>	<b>42</b>
3.8.1 Что такое WebSocket? .....	42
3.8.2 Преимущества WebSocket .....	42
3.8.3 Создание WebSocket-соединения .....	43
3.8.4 События.....	43
3.8.5 Отправка и получение данных.....	44
<b>3.9 Фреймворки для построения SPA .....</b>	<b>44</b>
3.9.1 Знакомство с Angular, React и Vue.....	44
3.9.2 Angular.....	44
3.9.3 React.....	45
3.9.4 Vue.js.....	45
3.9.5 Как выбрать между Angular, React и Vue? .....	46
<b>3.10 Продвинутые темы в JavaScript .....</b>	<b>46</b>
3.10.1 Модули и модули ES6 .....	46
3.10.2 Использование JavaScript для серверной разработки с Node.js.....	47
3.10.3 Использование Node.js в разных областях разработки.....	47
3.10.4 Зависимости.....	48
<b>3.11 Задание для самоподготовки.....</b>	<b>49</b>
3.11.1 Цель .....	49
3.11.2 Описание .....	49
3.11.3 Требования .....	50
3.11.4 Варианты.....	50
3.11.5 Контрольные вопросы.....	51
<b>4 Фреймворки JavaScript .....</b>	<b>53</b>
<b>4.1 Введение в фреймворки JavaScript .....</b>	<b>53</b>
4.1.1 Понятие фреймворка и его роль в веб-разработке .....	53

4.1.2 Преимущества использования фреймворков перед чистым JavaScript .....	53
4.1.3 Ограничения и недостатки фреймворков .....	53
<b>4.2 Основы TypeScript .....</b>	<b>54</b>
4.2.1 Введение в TypeScript .....	54
4.2.2 Типы данных и аннотации типов.....	54
4.2.3 Интерфейсы и классы в TypeScript.....	55
4.2.4 Преимущества TypeScript по сравнению с JavaScript.....	56
<b>4.3 Vue.js .....</b>	<b>56</b>
4.3.1 Установка и настройка .....	56
4.3.2 Основы компонентной архитектуры .....	57
4.3.3 Работа с директивами и шаблонами.....	59
<b>4.4 Задание для самоподготовки .....</b>	<b>60</b>
4.4.1 Задача.....	60
4.4.2 Требования.....	61
4.4.3 Варианты.....	61

# 1 HTML

## 1.1 Введение в HTML

### 1.1.1 Что такое HTML?

HTML, или "HyperText Markup Language" (язык гипертекстовой разметки), представляет собой фундаментальный язык веб-разработки, играющий ключевую роль в создании веб-страниц. HTML служит для определения структуры контента и способа его представления в веб-браузерах.

### 1.1.2 Зачем нужен HTML

HTML используется для разметки текстового и мультимедийного содержания в веб-документах. Этот язык определяет структуру информации, включая заголовки, абзацы, списки, таблицы, изображения и ссылки. Благодаря HTML браузеры могут корректно отображать веб-контент на экранах пользователей.

### 1.1.3 Краткая история HTML

HTML был разработан в начале 1990-х годов группой исследователей в ЦЕРН (Европейская организация по ядерным исследованиям) с целью облегчения обмена и публикации информации в научных кругах. Со временем HTML прошел через несколько версий и в настоящее время существует стандарт HTML5, который поддерживается большинством современных веб-браузеров. HTML5 предоставляет широкие возможности для создания интерактивных и удобочитаемых веб-страниц.

## 1.2 Основы HTML

### 1.2.1 Структура HTML-документа

HTML-документ имеет строго определенную структуру, которая включает в себя следующие основные элементы:

- `<!DOCTYPE>`: определяет версию HTML, с которой работает документ.

Например, `<!DOCTYPE html>` указывает на использование HTML5;



- `<html>`: обозначает начало и конец HTML-документа. Все содержимое веб-страницы должно находиться между открывающим и закрывающим тегами `<html>`;
- `<head>`: содержит метаинформацию о документе, такую как заголовок страницы, ссылки на внешние ресурсы (стили, скрипты), и другие метаданные;
- `<title>`: определяет заголовок веб-страницы, который отображается в строке заголовка браузера и является важным для поисковых систем;
- `<meta>`: Элемент `<meta>` используется для определения метатегов, которые предоставляют информацию о документе, такую как кодировка символов, описание, автор и другие метаданные;
- `<body>`: содержит основное содержимое веб-страницы, такое как текст, изображения, ссылки и другие элементы, которые отображаются в окне браузера.

```
<!DOCTYPE html>
<html>
<head>
  <title>Пример HTML-документа</title>
</head>
<body>
  <h1>Заголовок страницы</h1>
  <p>Это абзац текста.</p>
</body>
</html>
```

Рис. 1.1 – Пример HTML-документа

HTML-документ всегда начинается с объявления версии и типа документа и имеет корневой элемент `<html>`. Содержание страницы разделяется на секции, такие как `<head>` и `<body>`. Внутри секции `<head>` обычно находятся метаданные и заголовок страницы, а внутри `<body>` размещается видимый контент (Рис. 1.1).

### 1.2.2 Теги, элементы и атрибуты

HTML – это язык разметки, который работает на основе концепции тегов, элементов и атрибутов. Понимание этих основных понятий является важным шагом в освоении HTML:

- Теги – это основные строительные блоки HTML. Они обозначают начало и конец элемента на веб-странице. Теги обычно заключаются в угловые скобки, например, `<tag>`. Один из примеров тега - `<p>`, который используется для обозначения параграфов текста;
- элемент – это комбинация открывающего и закрывающего тега, а также содержимого, помещенного между этими тегами. Например, `<p>Это пример абзаца текста</p>` - здесь `<p>` - открывающий тег, `</p>` - закрывающий тег, а " Это пример абзаца текста " - содержимое элемента (Рис. 1.2);

A screenshot of a code editor with a dark background. It shows the HTML code `<p>Это пример абзаца текста.</p>` in a light-colored font. The opening and closing tags are highlighted in yellow.

Рис. 1.2 – Пример абзаца

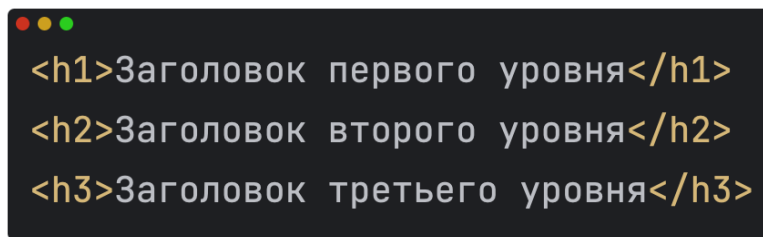
- атрибуты – это дополнительные сведения, которые можно присвоить элементам с помощью тегов. Атрибуты добавляют дополнительные характеристики к элементу. Например, атрибут `src` в теге `<img>` указывает источник изображения (Рис. 1.3).

A screenshot of a code editor with a dark background. It shows the HTML code `` in a light-colored font. The `src` attribute value is underlined with a wavy line, and the `alt` attribute value is highlighted in green.

Рис. 1.3 - Пример изображения

### 1.2.3 Заголовки

Заголовки используются для выделения заголовков разных уровней. Они начинаются с `<h1>` и заканчиваются `<h6>`, где `<h1>` - самый крупный заголовок, а `<h6>` - самый мелкий.



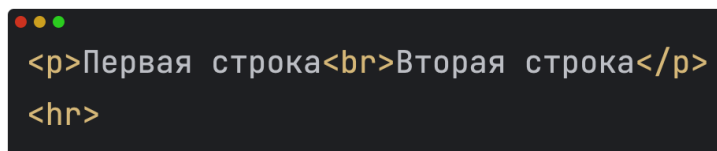
```
<h1>Заголовок первого уровня</h1>
<h2>Заголовок второго уровня</h2>
<h3>Заголовок третьего уровня</h3>
```

Рис. 1.4 – Пример заголовков

#### 1.2.4 Абзацы

Параграфы для создания абзацев используется тег `<p>`. Внутри тега `<p>` помещается текст, который будет отображаться как абзац (Рис. 1.2).

Строчные элементы используются для структурирования текста. Например, `<br>` используется для переноса текста на новую строку, а `<hr>` для создания горизонтальной линии.

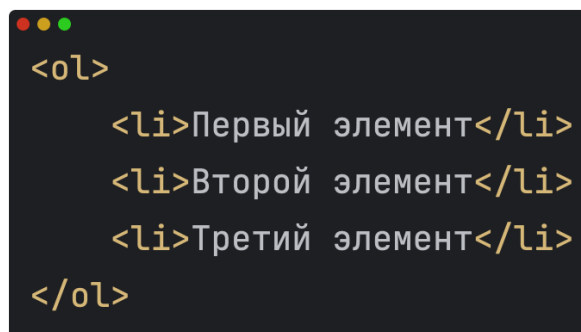


```
<p>Первая строка<br>Вторая строка</p>
<hr>
```

Рис. 1.5 – Пример строчных элементов

#### 1.2.5 Нумерованные списки

Нумерованные списки используются, когда важен порядок элементов, и каждый элемент должен быть пронумерован. Каждый элемент в нумерованном списке автоматически получает номер, начиная с 1.



```
<ol>
  <li>Первый элемент</li>
  <li>Второй элемент</li>
  <li>Третий элемент</li>
</ol>
```

Рис. 1.6 - Нумерованный список

### 1.2.6 Маркированные списки

Маркированные списки применяются, когда порядок элементов не имеет значения, и каждый элемент помечается маркером, таким как кружок, точка или другой символ.

```
<ul>
  <li>Элемент с кружком</li>
  <li>Элемент с точкой</li>
  <li>Другой элемент с кружком</li>
</ul>
```

Рис. 1.7 - Маркированный список

### 1.2.7 Списки определений

Списки определений – это специальный тип списка, который используется для создания списка терминов и их определений. В HTML для создания списков определений используются теги `<dl>` (description list) – представляет собой список описаний, `<dt>` (term) – список пар терминов, и `<dd>` (definition) – описание.

```
<dl>
  <dt>Термин 1</dt>
  <dd>Определение термина 1</dd>

  <dt>Термин 2</dt>
  <dd>Определение термина 2</dd>

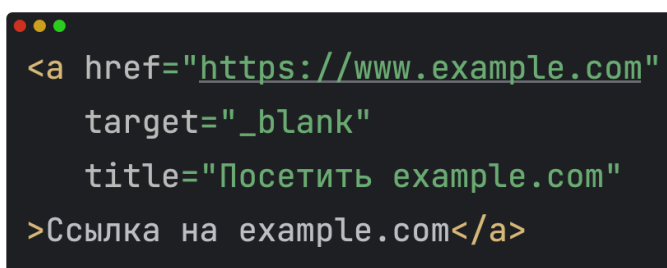
  <dt>Термин 3</dt>
  <dd>Определение термина 3</dd>
</dl>
```

Рис. 1.8 – Список определений

Списки определений особенно полезны для создания словарей и справочников на веб-страницах, где требуется представить термины и их значения.

### 1.2.8 Гиперссылки

Гиперссылки (или просто ссылки) – это ключевой элемент веб-страниц, который позволяет переходить с одной страницы на другую, а также переходить к другим ресурсам в интернете. В HTML для создания гиперссылок используется тег <a>. У данного тега есть несколько атрибутов:



```
<a href="https://www.example.com"
    target="_blank"
    title="Посетить example.com"
>Ссылка на example.com</a>
```

Рис. 1.9 - Пример ссылки

- href: Атрибут href указывает на адрес (URL) ресурса, на который ссылка ведет. Этот атрибут является обязательным для тега <a>;
- target: Атрибут target определяет, как будет открываться ссылка. Например, можно указать, чтобы ссылка открывалась в новом окне или в текущем окне;
- title: Атрибут title добавляет всплывающую подсказку, которая появляется при наведении указателя мыши на ссылку. Этот атрибут улучшает доступность и информативность ссылок;

Гиперссылки позволяют пользователям переходить между веб-страницами и взаимодействовать с разными ресурсами в интернете, делая их важным элементом веб-разработки.

### 1.2.9 Изображения

Изображения играют важную роль в веб-дизайне и могут значительно улучшить визуальное восприятие веб-страниц. Они используются для

отображения фотографий, иллюстраций, график и других визуальных элементов. В HTML изображения создаются с использованием элемента `<img>`.

```

```

Рис. 1.10 - Пример изображения

Тег `<img>` (image) используется для вставки изображений на веб-страницу. Этот тег является пустым, и он не имеет закрывающего тега. Вместо этого, для отображения изображения, используется атрибуты:

- src (source): Атрибут `src` задает путь к файлу изображения. Путь может быть относительным (относительно текущей директории веб-страницы) или абсолютным (полным URL-адресом изображения).
- alt (alternative text): Атрибут `alt` предоставляет альтернативный текст, который будет отображен, если изображение не может быть загружено или для пользователей с ограниченными возможностями (например, для чтения вслух программами). Альтернативный текст должен описывать содержание изображения.

### 1.2.10 Комментарии в HTML

Комментарии в HTML используются для вставки текстовых заметок или пояснений, которые не отображаются в браузере, но могут быть полезными для разработчиков и поддержки кода. Комментарии начинаются с `<!--` и заканчиваются `-->`. Все, что находится между этими символами, считается комментарием и игнорируется браузером.

Комментарии могут использоваться для объяснения структуры документа, временного исключения кода или любых других целей, где необходимо оставить пояснения.

### 1.2.11 Специальные символы и символы перевода строки

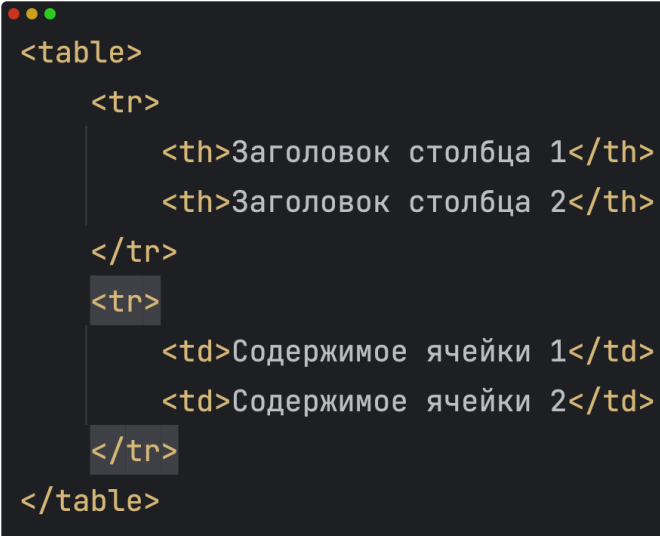
В HTML некоторые символы имеют специальное значение и не могут быть использованы напрямую в тексте. Вместо этого они должны быть заменены на соответствующие HTML-коды. Например:

- ☐ `&lt;`; заменяет символ `<`.
- ☐ `&gt;`; заменяет символ `>`.
- ☐ `&amp;`; заменяет символ `&`.
- ☐ `&quot;`; заменяет символ `"` (двойные кавычки).
- ☐ `&apos;`; заменяет символ `'` (апостроф).

Символы перевода строки, такие как `\n` или `<br>`, используются для создания переносов строки в тексте или разметке.

### 1.2.12 Таблицы

Таблицы в HTML используются для организации данных в удобном и структурированном виде. Они состоят из строк и столбцов и позволяют представить информацию в виде сетки.



```
<table>
  <tr>
    <th>Заголовок столбца 1</th>
    <th>Заголовок столбца 2</th>
  </tr>
  <tr>
    <td>Содержимое ячейки 1</td>
    <td>Содержимое ячейки 2</td>
  </tr>
</table>
```

Рис. 1.11 - Пример таблицы

Создание таблиц в HTML осуществляется с использованием следующих элементов:

- ☐ `<table>`: Элемент `<table>` определяет начало и конец таблицы. Все остальные элементы таблицы находятся между открывающим и закрывающим тегами `<table>`.

- `<tr>`: Элемент `<tr>` (table row) определяет строки таблицы. Каждая строка таблицы должна находиться между открывающим и закрывающим тегами `<table>`.
- `<td>`: Элемент `<td>` (table data) определяет ячейки данных внутри таблицы. Он размещается внутри строк (`<tr>`) и содержит сами данные.
- `<th>`: Элемент `<th>` (table header) используется для определения заголовков столбцов или строк таблицы. Он также размещается внутри строк (`<tr>`) и обычно выделяется жирным шрифтом.

### **1.3 Формы и элементы управления**

#### **1.3.1 Формы**

Формы в HTML позволяют пользователям отправлять данные на сервер для обработки. Они могут включать различные элементы управления, такие как текстовые поля, кнопки, переключатели, списки выбора и многие другие. Формы играют важную роль в интерактивности веб-страниц и сборе информации от пользователей.

Тег `<form>` используется для создания формы на веб-странице. Этот элемент определяет начало и конец формы, а также содержит другие элементы управления формой.



1.3.2 Текстовые поля, кнопки и переключатели

1.3.3 Списки выбора и множественный выбор

1.3.4 Текстовые области и элементы для загрузки файлов

1.3.5 Атрибуты форм и методы отправки данных

1.3.6 Валидация и атрибуты HTML5 для форм

## **1.4 Семантика и структура**

1.4.1 Семантические теги HTML5

1.4.2 <header>, <nav>, <main>, <article>, <section>

1.4.3 <aside>, <footer>, <figure>, <figcaption>

1.4.4 Заголовки и навигация

1.4.5 Аудио и видео

1.4.6 Теги <audio> и <video>

1.4.7 Атрибуты и поддерживаемые форматы

1.4.8 Ссылки и закладки

## **1.5 Задание для самоподготовки**

1.5.1 Цель

Целью этой работы является создание интерактивного веб-приложения для управления списком задач.

1.5.2 Описание

Необходимо создать веб-приложение, которое позволит пользователям управлять списком задач. Приложение должно иметь следующие функциональные возможности:

1. Добавление задачи: пользователь должен иметь возможность перейти на отдельную страницу и ввести там текст задачи, после чего нажать кнопку "Добавить".
2. Удаление задачи: каждая задача в списке должна иметь кнопку "Удалить".
3. Отметка задачи как выполненной: каждая задача должна иметь маркер (например, флажок), который позволяет пользователю отметить задачу как выполненную.

4. Список задач: должен быть создан с помощью тега table, в нем должно быть 3 колонки (задачи, в работе, выполненные).

### 1.5.3 Контрольные вопросы

1. Какой HTML-тег используется для создания формы ввода текста?
2. Какой атрибут HTML используется для создания текстового поля ввода в форме?
3. Как можно создать кнопку?
4. Какие HTML-теги используются для создания нумерованного списка?
5. Какие HTML-теги используются для создания маркированного списка?
6. Какой HTML-тег используется для создания таблицы?
7. Какой HTML-тег используется для определения заголовков столбцов в таблице?
8. Какой тег HTML используется для создания строки в таблице?
9. Какой тег HTML используется для создания ячейки в таблице?
10. Каким образом можно создать гиперссылку (ссылку) в HTML?
11. Какие HTML-теги используются для создания заголовков разного уровня?
12. Каким образом можно вставить изображение (графику) на веб-страницу с помощью HTML?
13. Как создать флажок (чекбокс) на веб-странице с помощью HTML?
14. Какой атрибут HTML используется для определения идентификатора элемента?
15. Какой тег HTML используется для создания заголовка страницы?
16. Какой атрибут HTML используется для определения адреса (URL) гиперссылки?

## **2 CSS**

### **2.1 Введение в CSS**

2.1.1 Что такое CSS и зачем оно нужно

2.1.2 Основные принципы работы CSS

2.1.3 Внедрение CSS в веб-страницу

### **2.2 Синтаксис CSS**

2.2.1 Селекторы

2.2.2 Свойства

2.2.3 Значения

2.2.4 Комментарии

### **2.3 Внедрение CSS в HTML**

2.3.1 Внутренние (встроенные) стили

2.3.2 Внешние таблицы стилей (external CSS)

2.3.3 Встроенные стили в HTML-элементах

### **2.4 Селекторы CSS**

2.4.1 Селекторы элементов

2.4.2 Селекторы классов

2.4.3 Селекторы идентификаторов

2.4.4 Комбинированные селекторы

### **2.5 Основные свойства и значения**

2.5.1 Цвет и фон

2.5.2 Текст и шрифты

2.5.3 Отступы и рамки

2.5.4 Позиционирование и размеры

## **2.6 Каскадность и наследование**

### 2.6.1 Приоритет селекторов

### 2.6.2 Каскадирование стилей

### 2.6.3 Наследование свойств

## **2.7 Практические примеры**

### 2.7.1 Создание навигационного меню

### 2.7.2 Оформление таблиц

### 2.7.3 Создание адаптивного дизайна

### 2.7.4 Работа с анимациями и переходами

## **2.8 Оптимизация и отладка**

### 2.8.1 Инструменты для отладки CSS

### 2.8.2 Минимизация и оптимизация кода

### 2.8.3 Работа с браузерными различиями

## **2.9 Расширенные темы**

### 2.9.1 Flexbox и Grid Layout

### 2.9.2 CSS-препроцессоры

### 2.9.3 CSS-фреймворки

## **2.10 Задание для самоподготовки**

### 2.10.1 Цель

Создать стили для интерактивного веб-приложения, которое управляет списком задач.

### 2.10.2 Описание

1. Создайте стили для заголовка (`<h1>`) веб-приложения. Укажите цвет текста, размер шрифта и выровняйте его по центру страницы.
2. Установите стили для формы добавления задачи (`<form>` и `<input>`). Укажите цвет фона, цвет текста, отступы и рамку для текстового поля ввода задачи. Сделайте кнопку "Добавить" (`<button>`) выделяющейся с помощью

цвета фона и текста. Сделайте так, чтобы форма и кнопка были выровнены по центру страницы.

3. Создайте стили для таблицы (`<table>`) и её заголовков (`<th>`). Установите цвета фона и текста для заголовков таблицы, а также добавьте небольшой отступ и выровняйте текст по центру.
4. Установите стили для строк таблицы и её ячеек (`<tr>` и `<td>`). Укажите цвета фона и текста, добавьте отступы и сделайте текст в ячейках выровненным по центру. Для ячеек с задачами, в работе и выполненными задачами, установите разные стили, чтобы их можно было легко отличать друг от друга.
5. Создайте стили для кнопок "Удалить" и маркера задачи (например, флажка). Сделайте кнопку "Удалить" выделяющейся и добавьте стиль для маркера, который будет изменяться, когда задача отмечена как выполненная.
6. Добавьте стили для hover-эффектов для кнопок "Удалить" и маркера задачи, чтобы они меняли свой внешний вид при наведении на них курсора.
7. Настройте стили так, чтобы приложение выглядело эстетично и было удобным в использовании для пользователя.

### 2.10.3 Варианты

1. Минималистичный стиль: создайте минималистичный стиль для веб-приложения. Используйте только черно-белую цветовую палитру. Сделайте фон страницы белым, текст черным. Уберите все лишние отступы и границы вокруг элементов. Задачи и кнопки должны быть простыми и четкими.
2. Яркий и цветной стиль: создайте яркий и цветной стиль для веб-приложения. Используйте яркие цвета для фона кнопок и маркеров задач. Добавьте анимацию для кнопок "Удалить" и маркеров при наведении курсора.
3. Стиль "Заметки": создайте стиль, который делает ваше приложение похожим на блокнот с заметками. Используйте текстурный фон (например, бумажный фон), выберите шрифт, который напоминает рукописный текст.

Добавьте тени к ячейкам таблицы, чтобы создать эффект слегка выпуклых заметок.

4. Стил "Пост-ит": создайте стил, который делает ваше приложение похожим на листки "пост-ит". Используйте разные яркие цвета для фона каждой задачи. Добавьте эффекты цветных маркеров (например, нижнее подчеркивание) для акцентирования задач.
5. Стил "Доска задач": создайте стил, который делает ваше приложение похожим на доску задач с карточками. Используйте карточки для каждой задачи с заголовком и текстом задачи. Добавьте эффекты тени к карточкам для трех разных столбцов (задачи, в работе, выполненные).

#### 2.10.4 Контрольные вопросы

1. Что такое CSS и для чего он используется в веб-разработке?
2. Какие два основных способа подключения CSS к HTML-документу?
3. Что такое селектор в CSS и какие виды селекторов вы знаете?
4. Как вы применяете встроенные стили в HTML-документе?
5. Что такое внешний CSS-файл и как его подключить к HTML-документу?
6. Какие атрибуты rel и type используются при подключении CSS-файла?
7. Какие единицы измерения используются в CSS для задания размеров и расстояний?
8. Что такое приоритеты в CSS и какие методы их определения вы знаете?
9. Какие элементы можно стилизовать с помощью CSS?
10. Что такое классы и идентификаторы в CSS, и в чем их различие?
11. Как создать комбинированный селектор в CSS?
12. Как изменить цвет текста с помощью CSS?
13. Как задать цвет фона элемента в CSS?
14. Как изменить шрифт текста с помощью CSS?
15. Что такое псевдо-классы и для чего они используются в CSS?
16. Как создать переходные эффекты (плавные изменения) с помощью CSS?
17. Как изменить размер и форму блока (div) с помощью CSS?
18. Как центрировать элемент по горизонтали и вертикали с помощью CSS?

19. Что такое позиционирование в CSS и какие значения свойства position вы знаете?
20. Как создать список с маркерами (bulleted list) с помощью CSS?
21. Как создать горизонтальное меню навигации с использованием CSS?
22. Что такое адаптивный дизайн (responsive design) и какие средства CSS используются для его создания?
23. Как скрыть элемент на веб-странице с помощью CSS?
24. Как изменить стиль ссылок (гиперссылок) в CSS, чтобы они выглядели по-разному в разных состояниях (наведение, активное состояние)?
25. Как создать таблицу стилей (CSS table) и какие стили можно применить к таблицам?
26. Как задать прозрачность (opacity) для элемента с помощью CSS?
27. Как создать анимацию с использованием CSS и какие свойства анимации вы можете настроить?
28. Как создать элемент с закругленными углами (rounded corners) с помощью CSS?
29. Как изменить порядок отображения элементов с помощью свойства z-index в CSS?
30. Какие CSS-препроцессоры вы знаете, и для чего они используются в веб-разработке?

## **3 Основы JavaScript**

### **3.1 Введение**

#### **3.1.1 Что такое JavaScript?**

JavaScript – это высокоуровневый, интерпретируемый язык программирования, который используется для создания интерактивных веб-сайтов. Этот язык был разработан Бренданом Айком в 1995 году и стал ключевой технологией для фронтенд-разработки.

#### **3.1.2 История развития JavaScript**

JavaScript был создан в компании Netscape и изначально назывался LiveScript. Однако, позднее, для стратегических целей, он был переименован в JavaScript. JavaScript стал стандартом ECMA[1] (European Computer Manufacturers Association) и был назван ECMAScript. В последние годы ECMAScript 6 (ES6) внес множество улучшений и новых функциональных возможностей в язык.

#### **3.1.3 Важность изучения JavaScript**

JavaScript играет ключевую роль в современной веб-разработке. Он позволяет создавать динамические и интерактивные веб-приложения. Изучение JavaScript открывает множество возможностей для разработчика, включая возможность создания SPA (одностраничных приложений), взаимодействия с сервером через AJAX и использования популярных фреймворков, таких как Angular, React и Vue.

### **3.2 Основы JavaScript**

#### **3.2.1 Синтаксис и правила написания кода**

Синтаксис – это набор правил, определяющих, как нужно писать код на языке программирования. В случае JavaScript, хорошее понимание синтаксиса является ключевым для правильного написания кода. Давайте рассмотрим основные аспекты синтаксиса и правила написания кода в JavaScript.



### 3.2.2 Комментарии

Комментарии позволяют добавлять пояснения и объяснения в коде. JavaScript поддерживает два вида комментариев:

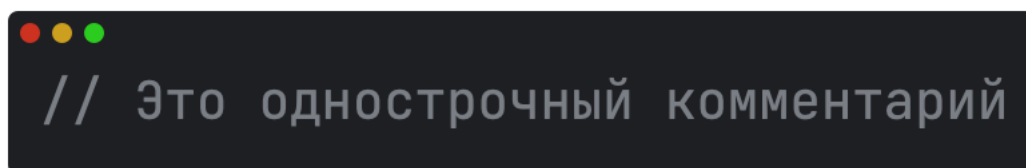
A screenshot of a code editor window with a dark background. At the top left, there are three colored window control buttons (red, yellow, green). The main area of the editor displays the text `// Это однострочный комментарий` in a light gray monospace font.

Рис. 3.1 – Однострочный комментарий

Однострочные комментарии, начинаются с `//` и продолжаются до конца строки (Рис. 3.1).

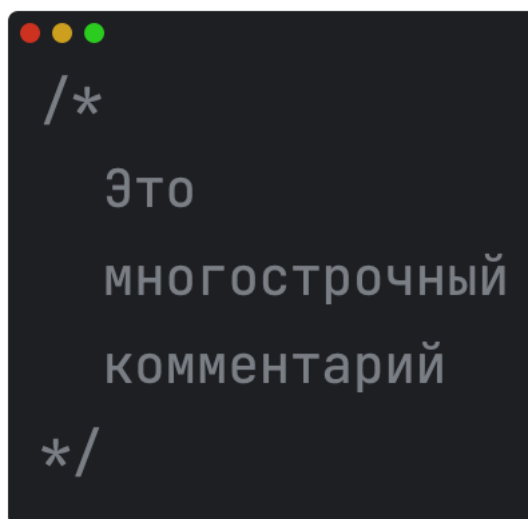
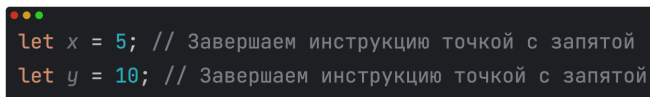
A screenshot of a code editor window with a dark background. At the top left, there are three colored window control buttons (red, yellow, green). The main area of the editor displays the text `/*  
 Это  
 многострочный  
 комментарий  
*/` in a light gray monospace font, with the text wrapped across four lines.

Рис. 3.2 – Многострочный комментарий

Многострочные комментарии, заключенные между `/*` и `*/`, могут охватывать несколько строк (Рис. 3.2).

### 3.2.3 Заключение инструкций в точку с запятой

В JavaScript инструкции обычно должны завершаться точкой с запятой (`;`). Это помогает интерпретатору понять, где заканчивается одна инструкция и начинается следующая.



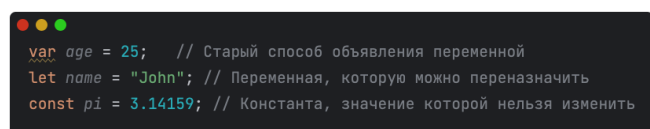
```
let x = 5; // Завершаем инструкцию точкой с запятой
let y = 10; // Завершаем инструкцию точкой с запятой
```

Рис. 3.3 – Пример окончания инструкции

Хотя в большинстве случаев точка с запятой необходима, JavaScript также способен автоматически вставлять точки с запятой в некоторых ситуациях, но надежнее явно указывать их.

### 3.2.4 Переменные

Переменные – это именованные контейнеры, в которых можно хранить данные. В JavaScript объявление переменной выполняется с использованием ключевых слов `var`, `let` или `const`.



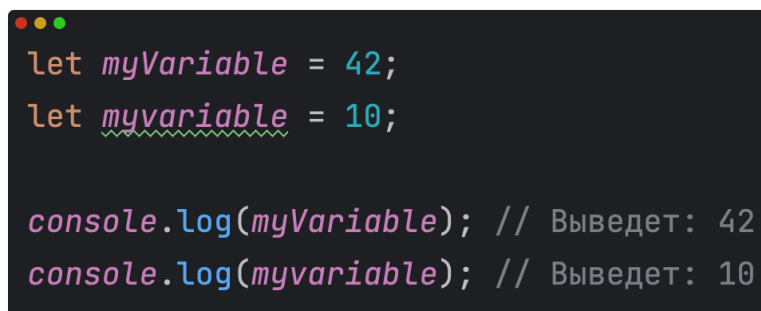
```
var age = 25; // Старый способ объявления переменной
let name = "John"; // Переменная, которую можно переназначить
const pi = 3.14159; // Константа, значение которой нельзя изменить
```

Рис. 3.4 – Примеры объявления переменных

Важно отметить различие между `let` и `const`. Переменные, объявленные с `let`, могут изменять свое значение, тогда как переменные, объявленные с `const`, являются константами и их значение нельзя изменить после присвоения.

### 3.2.5 Регистрозависимость

JavaScript является регистрозависимым языком, что означает, что он различает между строчными и заглавными буквами.



```
let myVariable = 42;
let myvariable = 10;

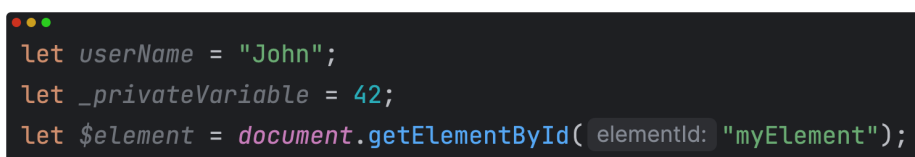
console.log(myVariable); // Выведет: 42
console.log(myvariable); // Выведет: 10
```

Рис. 3.5 – Пример регистрозависимости переменных

Переменные `myVariable` и `myvariable` будут считаться разными переменными.

### 3.2.6 Идентификаторы

Идентификаторы – это имена, которые используются для именования переменных, функций, объектов и других элементов в коде. Идентификаторы могут содержать буквы, цифры, символы подчеркивания `_` и знак доллара `$`. Они должны начинаться с буквы, символа подчеркивания `_` или знака доллара `$`.



```
let userName = "John";
let _privateVariable = 42;
let $element = document.getElementById("myElement");
```

Рис. 3.6 – Именование переменных

### 3.2.7 Типы данных

Типы данных – это характеристики данных, которые определяют их природу и как они могут быть обработаны. JavaScript поддерживает следующие основные типы данных:

Числа (Numbers): Целые числа и числа с плавающей точкой.

Строки (Strings): Текстовые данные, заключенные в кавычки (одинарные или двойные).

Булевы значения (Booleans): Логические значения `true` (истина) и `false` (ложь). Используются для условных выражений.

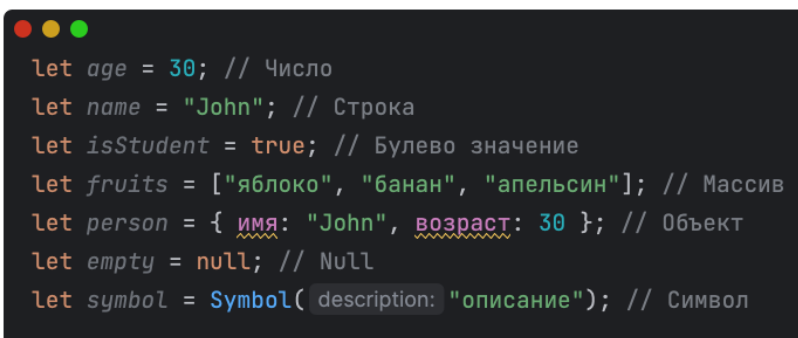
Массивы (Arrays): Упорядоченные списки элементов.

Объекты (Objects): Коллекции пар ключ-значение.

Undefined: Значение, которое имеет переменная, если ей не было присвоено никакого значения.

Null: Специальное значение, обозначающее "ничего" или "отсутствие значения".

Символы (Symbols): Уникальные и неизменяемые значения, используемые для создания свойств объектов.

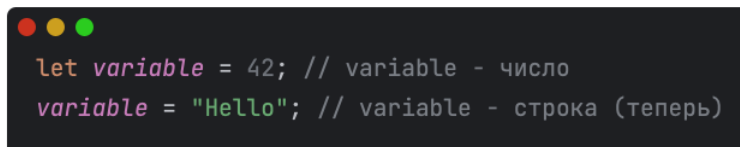


```
let age = 30; // Число
let name = "John"; // Строка
let isStudent = true; // Булево значение
let fruits = ["яблоко", "банан", "апельсин"]; // Массив
let person = { имя: "John", возраст: 30 }; // Объект
let empty = null; // Null
let symbol = Symbol(description: "описание"); // Символ
```

Рис. 3.7 – Примеры использования типов данных

### 3.2.8 Типизация

JavaScript является динамически типизированным языком, что означает, что тип переменной определяется автоматически во время выполнения, и можно присваивать переменным значения разных типов.

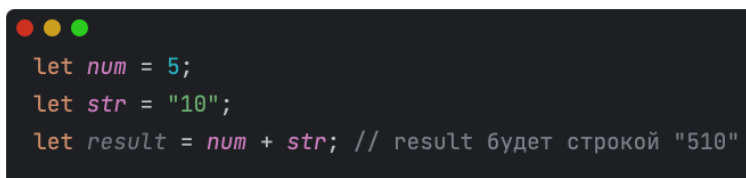


```
let variable = 42; // variable - число
variable = "Hello"; // variable - строка (теперь)
```

Рис. 3.8 – Пример динамической типизации

Тип данных переменной `variable` изменился из числа в строку.

Приведение типов (Type Coercion) - JavaScript также выполняет автоматическое приведение типов при выполнении операций с разными типами данных. Например, при сложении числа и строки, число может быть преобразовано в строку:



```
let num = 5;
let str = "10";
let result = num + str; // result будет строкой "510"
```

Рис. 3.9 – Приведение типов

### 3.2.9 Операторы и выражения

Операторы и выражения играют важную роль в JavaScript, так как они используются для выполнения действий, принятия решений и манипуляции данными в коде:

- **Арифметические операторы:** В JavaScript есть операторы для выполнения арифметических операций, таких как сложение (+), вычитание (-), умножение (\*), деление (/) и другие. Они используются для выполнения математических вычислений.
- **Операторы сравнения:** Эти операторы используются для сравнения значений. Например, оператор сравнения равенства (==) сравнивает два значения и возвращает true, если они равны, и false, если нет.
- **Логические операторы:** Операторы, такие как AND (&&), OR (||) и NOT (!), используются для создания сложных логических условий. Они часто применяются в условных выражениях.
- **Присваивания:** Операторы присваивания используются для присвоения значения переменной. Например, оператор "=" присваивает значение правой части выражения переменной слева.
- **Унарные операторы:** Унарные операторы применяются к одному операнду. Например, унарный минус (-) используется для изменения знака числа.
- **Операторы инкремента и декремента:** Операторы инкремента (++) и декремента (--) используются для увеличения или уменьшения значения переменной на 1.

Выражения – это комбинации операторов и операндов, которые выполняют конкретные вычисления. Выражения могут быть арифметическими, логическими, строковыми и т. д. Например, выражение  $2 + 3$  представляет собой арифметическое выражение, которое вернет результат 5. Выражения могут также включать переменные и вызовы функций.

### 3.2.10 Условные операторы и циклы

Условные операторы позволяют вам принимать решения и выполнять определенный блок кода, если условие истинно.

```
if (условие1) {  
    // Код для выполнения, если условие1 истинно  
} else if (условие2) {  
    // Код для выполнения, если условие1 ложно, а условие2 истинно  
} else {  
    // Код для выполнения, если ни одно из условий не истинно  
}
```

Рис. 3.10 - Пример условия

Оператор if выполняет код внутри блока, если указанное условие истинно. Оператор else if позволяет проверить несколько условий последовательно. Оператор else в свою очередь позволяет выполнить код, если условие в операторе if ложно.

Циклы позволяют выполнять блок кода многократно.

```
while (условие) {  
    // Код для выполнения, пока условие истинно  
}
```

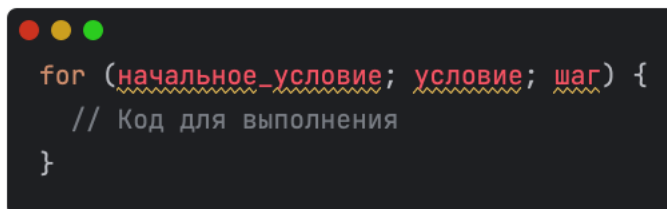
Рис. 3.11 - цикл с условием

Цикл while выполняет блок кода, пока указанное условие истинно.

```
do {  
    // Код для выполнения, хотя бы один раз  
} while (условие);
```

Рис. 3.12 - Цикл do...while

Цикл do...while выполняет блок кода, затем проверяет условие и повторяет выполнение, если условие истинно.



```

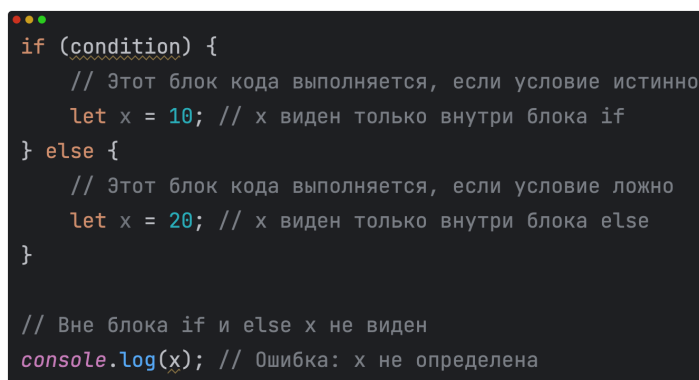
for (начальное_условие; условие; шаг) {
    // Код для выполнения
}

```

Рис. 3.13 - Цикл for

Цикл for используется для выполнения кода определенное количество раз.

### 3.2.11 Блоки кода



```

if (condition) {
    // Этот блок кода выполняется, если условие истинно
    let x = 10; // x виден только внутри блока if
} else {
    // Этот блок кода выполняется, если условие ложно
    let x = 20; // x виден только внутри блока else
}

// Вне блока if и else x не виден
console.log(x); // Ошибка: x не определена

```

Рис. 3.14 – Пример блока кода

Блоки кода обозначаются фигурными скобками {} и используются для группировки инструкций. Блоки кода определяют область видимости переменных (Рис. 3.14).

### 3.2.12 Функции и замыкания

Функции – это ключевой элемент JavaScript. Они позволяют упаковывать код в повторно используемые блоки и передавать аргументы для выполнения различных задач. Также важным концептом являются замыкания (Рис. 3.15), которые позволяют функциям иметь доступ к переменным из окружающей области видимости.



```
// Замыкания
1 usage
function createCounter() {
    let count : number = 0;
    return function () : void {
        count++;
        console.log(count);
    };
}

const counter = createCounter();
counter(); // 1
counter(); // 2
counter(); // 3
```

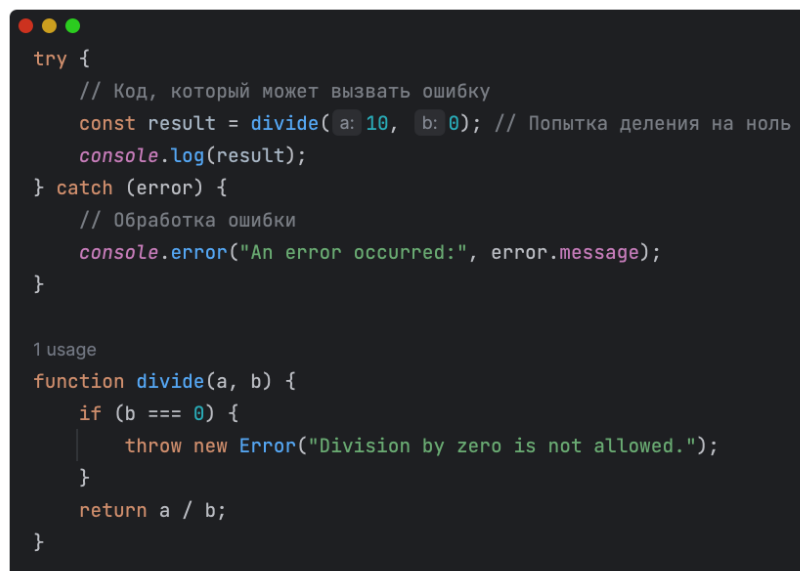
Рис. 3.15 – Работа замыкания

### 3.3 Работа с ошибками и отладка

#### 3.3.1 Обработка ошибок и исключений

Ошибки - неотъемлемая часть разработки программного обеспечения. В JavaScript есть механизм обработки ошибок и исключений с использованием оператора `try...catch` (Рис. 3.16). Этот механизм позволяет ловить ошибки и выполнять альтернативный код, если они возникают.





```

try {
    // Код, который может вызвать ошибку
    const result = divide(a: 10, b: 0); // Попытка деления на ноль
    console.log(result);
} catch (error) {
    // Обработка ошибки
    console.error("An error occurred:", error.message);
}

1 usage
function divide(a, b) {
    if (b === 0) {
        throw new Error("Division by zero is not allowed.");
    }
    return a / b;
}

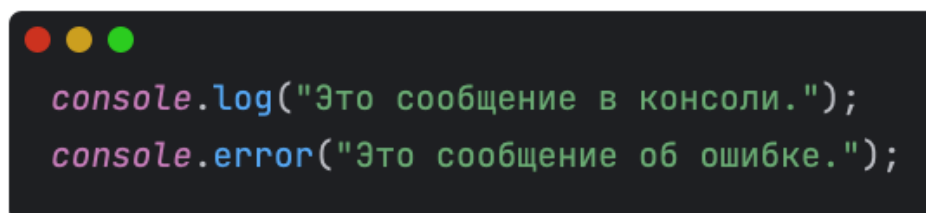
```

Рис. 3.16 – Обработка ошибок

### 3.3.2 Отладка с помощью инструментов разработчика

Отладка – это процесс поиска и исправления ошибок в коде. Веб-браузеры предоставляют мощные инструменты для отладки JavaScript-кода. Ниже приведены некоторые из них:

**Консоль (Console):** Вкладка "Console" в инструментах разработчика позволяет выводить сообщения, переменные и ошибки. Это полезно для отладки и вывода информации о коде.



```

console.log("Это сообщение в консоли.");
console.error("Это сообщение об ошибке.");

```

Рис. 3.17 – Вывод сообщений в консоль

**Точки останова (Breakpoints):** Можно устанавливать точки останова в коде, чтобы приостанавливать выполнение программы на определенной строке. Это позволяет анализировать состояние переменных и шаг за шагом выполнять код.

Инспектирование переменных (Variable Inspection): Можно просматривать значения переменных в текущем контексте выполнения и отслеживать их изменения во время отладки.

Стек вызовов (Call Stack): Вкладка "Call Stack" показывает текущий стек вызовов функций. Это помогает понять, какие функции вызываются и в каком порядке.

Инструменты для анализа кода (Sources): В разделе "Sources" можно просматривать исходный код страницы, устанавливать точки останова и анализировать выполнение кода.

Сетевые запросы (Network): Вкладка "Network" позволяет отслеживать сетевые запросы и анализировать их, что особенно полезно при отладке AJAX-запросов к серверу.

Внимание к этим инструментам и их использование помогают эффективно находить и устранять ошибки.

### 3.3.3 Типичные ошибки

При разработке JavaScript-приложений часто возникают типичные ошибки. Некоторые из них включают:

**TypeError:** Ошибка, возникающая, когда вы пытаетесь выполнить операцию с несовместимыми типами данных.

**ReferenceError:** Ошибка, возникающая, когда вы обращаетесь к неопределенной переменной.

**SyntaxError:** Ошибка, возникающая, когда код содержит синтаксическую ошибку.

## 3.4 Работа с DOM и HTML

### 3.4.1 Основы работы с DOM

DOM (Document Object Model) представляет собой иерархическую структуру, которая представляет веб-страницу в виде дерева объектов. Элементы HTML, такие как теги `<div>`, `<p>`, `<h1>` и другие, представляются в виде объектов в этой модели. JavaScript позволяет вам взаимодействовать с этой моделью, изменяя содержимое и структуру веб-страницы.

### 3.4.2 Создание и манипулирование элементами HTML

JavaScript позволяет создавать новые HTML-элементы и изменять существующие. Это делается с помощью методов, таких как `createElement()`, `appendChild()`, `removeChild()` и других.

```
// Создание нового элемента
const newDiv : HTMLDivElement = document.createElement( tagName: "div");

// Добавление текстового содержимого
newDiv.textContent = "Это новый элемент div";

// Добавление элемента в DOM
const container : HTMLElement = document.getElementById( elementId: "container");
container.appendChild(newDiv);
```

Рис. 3.18 – Создание и добавление элемента в DOM

### 3.4.3 Манипуляция стилями и классами

JavaScript также позволяет вам изменять стили элементов и управлять их классами. Это полезно для создания анимаций и изменения внешнего вида веб-страницы в ответ на действия пользователя.

```
const element : HTMLElement = document.getElementById( elementId: "myElement");

// Изменение цвета фона
element.style.backgroundColor = "blue";

// Добавление класса
element.classList.add("highlight");
```

Рис. 3.19 – Изменение стилей элемента

### 3.4.4 События

События – это действия, которые происходят на веб-странице, такие как клик мыши, нажатие клавиши, загрузка ресурсов и многие другие. JavaScript позволяет прослушивать и реагировать на различные события, что делает веб-страницы интерактивными и динамичными.

Вот некоторые из наиболее часто используемых событий:

`click`: Событие происходит при клике на элементе.

`mouseover` и `mouseout`: События происходят, когда курсор мыши наводится на элемент и уходит с него.

keydown и keyup: События происходят при нажатии и отпуске клавиши на клавиатуре.

submit: Событие происходит при отправке формы.

load: Событие происходит, когда элемент (например, изображение) полностью загружен.

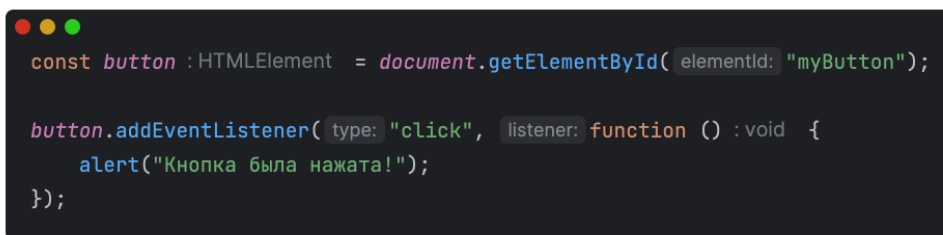
resize: Событие происходит при изменении размеров окна браузера.

scroll: Событие происходит при прокрутке содержимого элемента (например, страницы).

focus и blur: События происходят, когда элемент получает и теряет фокус ввода.

### 3.4.5 Обработка событий в JavaScript

Для прослушивания событий и выполнения определенных действий в ответ на них, вы можете использовать метод `addEventListener()`. Этот метод принимает два аргумента: тип события и функцию-обработчик.



```
const button :HTMLInputElement = document.getElementById( "myButton");

button.addEventListener( type: "click", listener: function () :void {
    alert("Кнопка была нажата!");
});
```

Рис. 3.20 – Добавление обработчика события клика

В этом примере (Рис. 3.20), мы добавляем обработчик события `click` к кнопке с идентификатором `"myButton"`. Когда пользователь кликает на эту кнопку, выполняется указанная функция.

Иногда бывает необходимо удалить обработчик события, например, если вам больше не нужно реагировать на определенное событие. Для этого используется метод `removeEventListener()`. Однако для удаления обработчика, необходимо использовать ту же функцию-обработчик, которая была добавлена ранее. Поэтому, функции-обработчики должны быть именованными, чтобы их можно было корректно удалить.

```

const button : HTMLInputElement = document.getElementById( elementId: "myButton");

button.addEventListener( type: "click", handleClick);

// Для удаления обработчика, используем ту же функцию-обработчик
button.removeEventListener( type: "click", handleClick);

```

Рис. 3.21 – Удаление обработчика события клика

### 3.4.6 Объект события (Event Object)

Когда событие происходит, браузер создает объект события, который содержит информацию о событии и передает его в функцию-обработчик. Объект события часто называется event (или любым другим именем, которое вы укажете при объявлении функции-обработчика).

```

const button : HTMLInputElement = document.getElementById( elementId: "myButton");

button.addEventListener( type: "click", listener: function (event : MouseEvent ) : void {
    alert("Кнопка была нажата! Тип события: " + event.type);
});

```

Рис. 3.22 – Получение информации о событии

В этом примере (Рис. 3.22) мы используем объект события event для получения информации о типе события (event.type). Объект события также содержит другие полезные свойства, такие как event.target (ссылка на элемент, который вызвал событие) и многое другое.

## 3.5 Объектная модель браузера (BOM)

### 3.5.1 Основные понятия BOM

Объектная модель браузера (BOM) представляет собой набор объектов и методов JavaScript, которые позволяют взаимодействовать с браузером, его окном и другими элементами, не связанными с веб-страницей. BOM включает в себя объекты, такие как window, document, navigator, location и другие.

### 3.5.2 Использование BOM для взаимодействия с браузером

BOM объекты доступны через глобальный объект window. BOM предоставляет множество функциональных возможностей для взаимодействия с браузером:

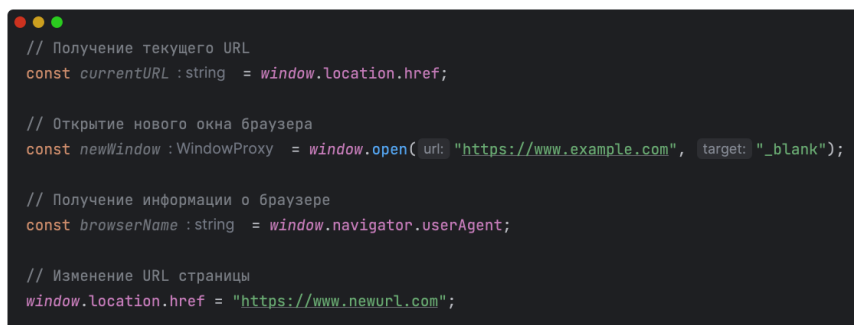
управление окнами и вкладками: Вы можете открывать новые окна и вкладки, закрывать их, а также управлять их размерами и положением;

Работа с историей браузера: Вы можете добавлять и удалять записи из истории браузера и переходить между страницами.

Управление браузерными событиями: BOM позволяет обрабатывать события, такие как загрузка страницы, закрытие окна и другие.

Работа с cookie и хранилищами: Вы можете управлять cookie, Local Storage и Session Storage для хранения данных на стороне клиента.

Геолокация: BOM предоставляет доступ к данным о местоположении устройства пользователя.



```
// Получение текущего URL
const currentURL :string = window.location.href;

// Открытие нового окна браузера
const newWindow :WindowProxy = window.open( url: "https://www.example.com", target: "_blank");

// Получение информации о браузере
const browserName :string = window.navigator.userAgent;

// Изменение URL страницы
window.location.href = "https://www.newurl.com";
```

Рис. 3.23 – Доступ к различным BOM объектам

## 3.6 Взаимодействие с сервером и AJAX

### 3.6.1 Основы AJAX

AJAX (Asynchronous JavaScript and XML) – это технология, позволяющая выполнять асинхронные запросы к серверу и обновлять части веб-страницы без перезагрузки всей страницы. AJAX использует объект XMLHttpRequest или методы, предоставляемые современными браузерами, такие как fetch, для отправки HTTP-запросов к серверу.

### 3.6.2 XMLHttpRequest и его методы

XMLHttpRequest – это объект, который предоставляет возможность отправки HTTP-запросов и получения ответов от сервера. Он имеет ряд методов для управления запросами.



```

const xhr : XMLHttpRequest = new XMLHttpRequest();
xhr.open( method: "GET", url: "https://api.example.com/data", async: true);

xhr.onload = function () :void {
    if (xhr.status === 200) {
        const responseData = JSON.parse(xhr.responseText);
        console.log(responseData);
    }
};

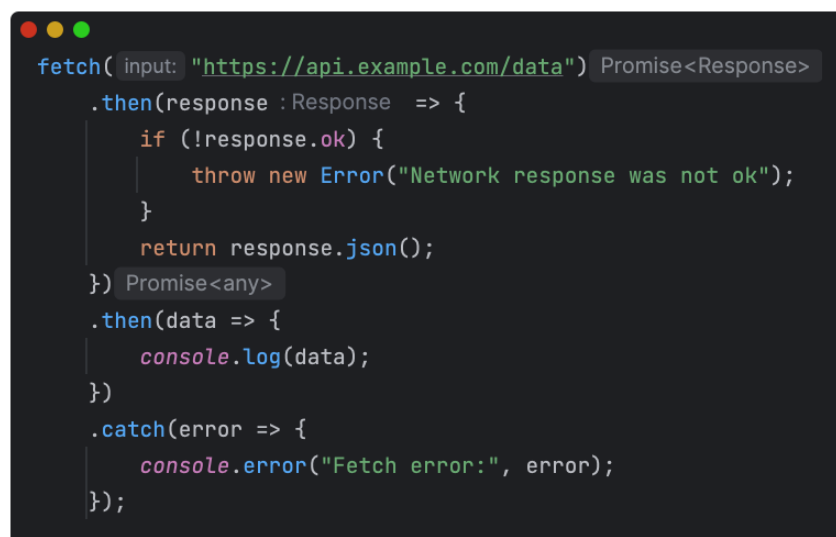
xhr.send();

```

Рис. 3.24 – Отправка GET-запроса с использованием XMLHttpRequest

### 3.6.3 Fetch API

fetch – это современный API, который предоставляет более удобный способ отправки HTTP-запросов и обработки ответов. Он возвращает обещание (Promise), что упрощает асинхронный код.



```

fetch( input: "https://api.example.com/data") Promise<Response>
    .then(response : Response => {
        if (!response.ok) {
            throw new Error("Network response was not ok");
        }
        return response.json();
    }) Promise<any>
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.error("Fetch error:", error);
    });

```

Рис. 3.25 – Отправка GET-запроса с использованием Fetch API

### 3.6.4 JSON: передача и обработка данных

JSON (JavaScript Object Notation) – это формат данных, который часто используется для обмена данными между клиентом и сервером. В JavaScript объекты можно легко преобразовывать в формат JSON и обратно с помощью методов `JSON.stringify()` и `JSON.parse()`.

```
const data : {age: number, name: string} = { name: "John", age: 30 };

// Преобразование объекта в JSON
const jsonData : string = JSON.stringify(data);

// Преобразование JSON в объект
const parsedData = JSON.parse(jsonData);
```

Рис. 3.26 – Преобразование объекта в JSON и обратно

JSON часто используется при работе с серверами и API для передачи структурированных данных.

### 3.6.5 Плюсы и минусы использования AJAX

Плюсы:

1. Асинхронность: AJAX позволяет загружать данные без перезагрузки всей страницы, что делает пользовательский опыт более плавным.
2. Эффективность: вы можете загружать только те данные, которые необходимы для обновления страницы, что экономит трафик и ускоряет загрузку.
3. Интерактивность: при помощи AJAX можно создавать интерактивные элементы, такие как автозаполнение форм, динамически обновляемые списки и другие.

Минусы:

1. Обработка ошибок: необходимо аккуратно обрабатывать ошибки, так как асинхронные запросы могут завершиться неудачно.
2. Безопасность: неправильно настроенные AJAX-запросы могут быть использованы для атак, таких как CSRF (межсайтовая подделка запросов).
3. SEO: Поисковые системы могут иметь проблемы с индексацией контента, загружаемого с помощью AJAX, если он не настроен правильно.



## 3.7 Работа с формами и валидация данных

### 3.7.1 Взаимодействие с разными типами форм

Формы являются важной частью веб-страниц и позволяют пользователям отправлять данные на сервер. JavaScript позволяет вам взаимодействовать с формами, получать данные, изменять их и отправлять обратно на сервер.

```
const inputElement : HTMLInputElement = document.getElementById( elementId: "username");
const username = inputElement.value;
```

Рис. 3.27 – Получение значения поля ввода формы

### 3.7.2 Валидация входных данных

Валидация данных – это проверка данных, вводимых пользователем, на соответствие определенным правилам или шаблонам. Валидация ввода на стороне клиента помогает предотвратить отправку некорректных данных на сервер и улучшить пользовательский опыт.

```
const emailInput : HTMLInputElement = document.getElementById( elementId: "email");
const email = emailInput.value;

const emailPattern : RegExp = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;

if (!emailPattern.test(email)) {
    alert("Введите некорректный email-адрес.");
}
```

Рис. 3.28 – Валидация email-адреса

### 3.7.3 Отправка данных на сервер

После валидации и обработки данных, вы можете отправить их на сервер с помощью AJAX-запроса или с использованием стандартной отправки формы.

```

const form : HTMLFormElement = document.getElementById( "myForm" );

form.addEventListener( type: "submit", listener: function ( event : SubmitEvent ) : void {
    event.preventDefault(); // Отмена стандартной отправки формы

    // Получение данных из формы
    const formData : FormData = new FormData(form);

    // Отправка данных на сервер
    fetch( input: "https://api.example.com/submit", init: {
        method: "POST",
        body: formData,
    })
        .then(response : Response => {
            if (response.ok) {
                alert("Данные успешно отправлены на сервер.");
            } else {
                alert("Произошла ошибка при отправке данных.");
            }
        })
        .catch(error => {
            alert("Произошла ошибка при отправке данных: " + error.message);
        });
});

```

Рис. 3.29 – Отправка формы на сервер

## 3.8 Websocket

### 3.8.1 Что такое WebSocket?

WebSocket - это протокол для двусторонней связи между клиентом и сервером через одно и то же соединение. Протокол обеспечивает реальное время обмена данными между клиентом и сервером.

### 3.8.2 Преимущества WebSocket

- Низкая задержка: websocket обеспечивает минимальную задержку при передаче данных;
- двунаправленное взаимодействие: клиент и сервер могут отправлять данные друг другу в любой момент;
- эффективность ресурсов: websocket использует меньше ресурсов сервера и сетевой пропускной способности, чем многие другие методы обмена данными.

### 3.8.3 Создание WebSocket-соединения

Для создания WebSocket-соединения на клиентской стороне, можно использовать конструктор WebSocket.

```
const socket : WebSocket = new WebSocket('ws://example.com');

socket.send('Отправить данные на сервер');

socket.addEventListener('message', (event : WebSocket.MessageEvent) : void => {
  console.log('Получено сообщение: ' + event.data);
});
```

Рис. 3.30 - Создание WS-соединения и обработка данных на клиентской стороне

Для создания WebSocket-сервера требует использования специализированных библиотек, таких как ws для Node.js.

```
import WebSocket from 'ws';

const server : WebSocket.Server<WebSocket, In... = new WebSocket.Server({ port: 8080 });

server.on('connection', (socket : WebSocket) : void => {
  console.log('Новое подключение');

  socket.on('message', (message : WebSocket.RawData) : void => {
    // Обработка сообщений от клиента
    socket.send('Привет, клиент!');
  });

  socket.on('close', () : void => {
    console.log('Соединение закрыто');
  });
});
```

Рис. 3.31 - Создание WebSocket-сервера на Node.js

### 3.8.4 События

WebSocket генерирует несколько событий, которые могут быть обработаны как на клиентской, так и на серверной стороне.

- ☐ open: срабатывает, когда соединение установлено;
- ☐ message: срабатывает, когда данные получены от другой стороны (клиента или сервера);
- ☐ close: срабатывает, когда соединение закрыто;
- ☐ error: срабатывает, если произошла ошибка.

### 3.8.5 Отправка и получение данных

Для отправки данных на сервер, используется метод `send` (Рис. 3.30). Для обработки данных, полученных от сервера, используется обработчик события `message` (Рис. 3.30).

На сервере, чтобы отправить данные клиенту, используется метод `send` на объекте сокета (Рис. 3.31). Так же есть возможность отправки данных конкретному клиенту.

## 3.9 Фреймворки для построения SPA

### 3.9.1 Знакомство с Angular, React и Vue

SPA (Single Page Application) – это веб-приложение, которое загружает только одну веб-страницу и динамически обновляет её содержимое без перезагрузки страницы. Для разработки SPA часто используются JavaScript-фреймворки и библиотеки.

#### 3.9.2 Angular

Angular – это полноценный фреймворк для разработки SPA и динамических веб-приложений. Он предоставляет все необходимые инструменты для создания крупных и сложных проектов, включая управление состоянием, маршрутизацию, средства тестирования и многие другие.



```
import { Component } from '@angular/core';

no usages
@Component({
  selector: 'app-root',
  template: `
    <h1>{{ title }}</h1>
    <button (click)="increment()">Увеличить</button>
    <p>Счетчик: {{ count }}</p>
  `,
})
export class AppComponent {
  title : string = 'Пример с Angular';
  count : number = 0;

  no usages
  increment() : void {
    this.count++;
  }
}
```

Рис. 3.32 – Пример использования Angular

### 3.9.3 React

React – это библиотека для создания пользовательских интерфейсов и компонентов. Он хорошо подходит для разработки как маленьких, так и больших SPA, а также для встраивания компонентов в существующие проекты.

```
import React, { useState } from 'react';

1 usage
function App() {
  const [count, setCount] = useState( initialState: 0);

  return (
    <div>
      <h1>Пример с React</h1>
      <button onClick={() : void => setCount( value: count + 1)}>Увеличить</button>
      <p>Счетчик: {count}</p>
    </div>
  );
}

no usages
export default App;
```

Рис. 3.33 – Пример использования React

### 3.9.4 Vue.js

Vue.js – это прогрессивный фреймворк, который может использоваться как для создания простых интерактивных элементов на странице, так и для разработки полноценных SPA. Он также легко внедряется в существующие проекты.

```
<template>
  <div>
    <h1>Пример с Vue.js</h1>
    <button @click="increment">Увеличить</button>
    <p>Счетчик: {{ count }}</p>
  </div>
</template>

<script>
no usages
export default {
  data() {
    return {
      count: 0,
    };
  },
  methods: {
    increment() {
      this.count++;
    },
  },
};
</script>
```

### Рис. 3.34 – Пример использования Vue.js

#### 3.9.5 Как выбрать между Angular, React и Vue?

Выбор фреймворка зависит от конкретных потребностей проекта и опыта. Вот некоторые соображения:

- Angular подходит для больших и сложных проектов, где требуется полная инфраструктура и управление состоянием приложения. Также Angular предоставляет множество инструментов для разработки.

React хорош для небольших и средних проектов, а также для интеграции в существующие приложения. Он сосредотачивается на компонентном подходе и переиспользовании кода.

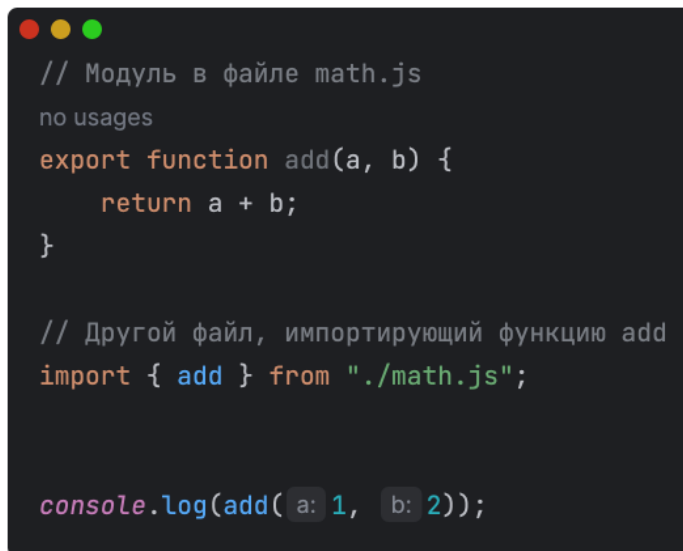
Vue.js подходит для разработки небольших и средних проектов, а также для быстрого создания интерактивных элементов. Он обладает простым API и легко внедряется в проекты.

Важно учитывать, что выбор фреймворка также может зависеть от предпочтений разработчиков и особенностей проекта.

### 3.10 Продвинутые темы в JavaScript

#### 3.10.1 Модули и модули ES6

Модули в JavaScript позволяют организовать код в отдельные файлы и модули, что улучшает его организацию и обеспечивает возможность повторного использования. Стандарт ES6 (ECMAScript 2015) ввел систему модулей, которая стала широко используемой в современных веб-приложениях.



```
// Модуль в файле math.js
no usages
export function add(a, b) {
  return a + b;
}

// Другой файл, импортирующий функцию add
import { add } from './math.js';

console.log(add(1, 2));
```

Рис. 3.35 – Пример использования модулей ES6

### 3.10.2 Использование JavaScript для серверной разработки с Node.js

Node.js – это среда выполнения JavaScript на стороне сервера, которая позволяет создавать серверные приложения и веб-сервисы. Она идеально подходит для создания высокоэффективных и масштабируемых серверных приложений.



```
import {createServer} from 'http';

const server = createServer( requestListener: (req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, World!\n');
});

const port = 3000;
server.listen(port, hostname: () => {
  console.log('Сервер работает на порту ${port}');
});
```

Рис. 3.36 – JavaScript для серверной разработки с Node.js

### 3.10.3 Использование Node.js в разных областях разработки

Node.js не ограничивается только серверным программированием. Вот некоторые области, где также можно использовать Node.js:

1. Разработка веб-приложений: Node.js позволяет создавать как серверные, так и клиентские части веб-приложений.

2. Инструменты командной строки: Node.js может быть использован для создания мощных инструментов командной строки, автоматизации задач, обработки файлов и других задач, связанных с управлением системой.
3. Разработка десктопных приложений: С использованием фреймворков, таких как Electron и NW.js, Node.js может быть использован для создания кроссплатформенных десктопных приложений.
4. Разработка игр: Node.js может быть использован для разработки онлайн и многопользовательских игр, как на серверной стороне, так и для создания клиентской части.
5. Интернет вещей (IoT): Node.js может быть использован для разработки программного обеспечения для устройств IoT благодаря его небольшому размеру и низкому потреблению ресурсов.
6. Разработка API и микро сервисов: Node.js хорошо подходит для создания API и микро сервисов благодаря высокой производительности и возможности обработки большого количества одновременных запросов.
7. Интерактивные приложения: Node.js может использоваться для создания интерактивных приложений, таких как чат-боты, игры в реальном времени и другие приложения, где важна мгновенная реакция на действия пользователей.

Таким образом, Node.js — это универсальная среда выполнения JavaScript, которая может быть применена в разных областях разработки, помимо серверных приложений.

#### 3.10.4 Зависимости

Зависимостью представляют собой модули, библиотеки или фреймворки, которые могут быть использованы в Node.js-приложениях. Каждый пакет содержит JavaScript-код и метаданные, такие как имя, версия, авторы, зависимости и описание.

npm (Node Package Manager) — это инструмент управления зависимостями и пакетами для языка JavaScript. Он является одним из наиболее распространенных и полезных инструментов в экосистеме Node.js.



npm предоставляет доступ к библиотекам, модулям и фреймворкам JavaScript, а также управляет зависимостями вашего проекта.

Основные концепции npm:

1. Пакеты (Packages): библиотеки, модули и фреймворки JavaScript, которые можно устанавливать и использовать в проектах. Каждый пакет имеет уникальное имя и версию.
2. package.json: файл конфигурации вашего проекта, который содержит информацию о проекте, его зависимостях и скриптах для управления проектом.

### **3.11 Задание для самоподготовки**

#### **3.11.1 Цель**

Целью этой работы является создание интерактивного веб-приложения для управления списком задач. Приложение должно позволять пользователю добавлять, удалять и отмечать задачи как выполненные.

#### **3.11.2 Описание**

Необходимо создать веб-приложение, которое позволит пользователям управлять списком задач. Приложение должно иметь следующие функциональные возможности:

1. Добавление задачи: Пользователь должен иметь возможность ввести текст задачи и нажать кнопку "Добавить", чтобы добавить новую задачу в список. Задача должна отображаться в списке задач.
2. Удаление задачи: Каждая задача в списке должна иметь кнопку "Удалить", которая позволяет пользователю удалить задачу из списка.
3. Отметка задачи как выполненной: Каждая задача должна иметь маркер (например, флажок), который позволяет пользователю отметить задачу как выполненную. Завершенные задачи должны быть выделены другим стилем или иметь другой визуальный индикатор.

4. Сохранение задач: Сделайте так, чтобы задачи сохранялись между сеансами. После обновления страницы или закрытия браузера задачи должны оставаться в списке.

### 3.11.3 Требования

- ☐ Используйте HTML, CSS и JavaScript для создания веб-приложения.
- ☐ Примените принципы модульного программирования и разделите ваш код на модули (например, модуль для работы с задачами, модуль для пользовательского интерфейса).
- ☐ Обеспечьте грамотное управление состоянием приложения, чтобы изменения в данных отражались на интерфейсе и наоборот.
- ☐ Дизайн приложения должен быть чистым и интуитивно понятным для пользователя.
- ☐ Используйте локальное хранилище браузера (localStorage) для сохранения задач между сеансами.

### 3.11.4 Варианты

1. Добавьте возможность редактировать текст задачи после её создания.
2. Реализуйте сортировку задач по алфавиту (по тексту) в обоих направлениях (по возрастанию и убыванию).
3. Добавьте возможность устанавливать срок выполнения для каждой задачи и отображать оставшееся время.
4. Разделите задачи на категории (например, работа, личное, учеба) и добавьте фильтрацию по категориям.
5. Создайте функцию поиска, которая позволяет пользователю быстро находить задачи по ключевым словам.
6. Добавьте возможность устанавливать приоритет для задач (например, низкий, средний, высокий) и сортировать задачи по приоритету.
7. Внедрите многопользовательский режим, где каждый пользователь имеет свой список задач и может входить в систему.
8. Реализуйте историю изменений задач, чтобы пользователь мог просматривать и восстанавливать предыдущие версии.

9. Добавьте возможность экспорта и импорта списка задач в файлы для резервного копирования и обмена данными.
10. Реализуйте тегирование задач (добавление ключевых слов) и возможность фильтрации по тегам.
11. Добавьте возможность комментирования задач и ведения обсуждений.
12. Реализуйте режим "Темная тема" для приложения с возможностью переключения между темами.
13. Добавьте функцию "Перетащить и бросить" для сортировки задач перетаскиванием.
14. Добавьте возможность экспортировать данные в формате CSV.
15. Интегрируйте приложение с календарем, чтобы пользователи могли видеть свои задачи на календаре добавив возможность экспортировать данные в формате ICS.

### 3.11.5 Контрольные вопросы

1. Что такое JavaScript?
2. Как создать переменную в JavaScript?
3. Какие типы данных поддерживает JavaScript?
4. Как объявить константу в JavaScript?
5. Как объявить функцию в JavaScript?
6. Какие различия между var, let и const в объявлении переменных?
7. Что такое область видимости переменных (scope) в JavaScript?
8. Какие типы данных относятся к ссылочным (non-primitive) в JavaScript?
9. Какие условные операторы существуют в JavaScript?
10. Как создать условие если в JavaScript?
11. Что такое оператор "switch" и как он используется?
12. Как создать цикл "for" в JavaScript?
13. Как создать цикл "while" в JavaScript?
14. Как можно завершить выполнение цикла преждевременно?
15. Как можно перейти к следующей итерации цикла?
16. Какие параметры можно передавать в функцию в JavaScript?

17. Что такое область видимости переменных в функциях?
18. Как объявить анонимную функцию (лямбда-функцию) в JavaScript?
19. Какие различия между функциональными выражениями и объявлениями функций?
20. Что такое замыкание (closure) в JavaScript?
21. Как можно передавать функцию как аргумент другой функции?
22. Как создать массив в JavaScript?
23. Как добавить элемент в массив?
24. Как удалить элемент из массива?
25. Как получить длину массива?
26. Что такое объект в JavaScript?
27. Как создать объект и добавить свойства и методы?
28. Как получить доступ к свойствам и методам объекта?
29. Как назначить обработчик события элементу в DOM?
30. Какие типы событий существуют в JavaScript?
31. Как предотвратить стандартное поведение элемента при срабатывании события?
32. Как передать параметры в обработчик события?
33. Как удалить обработчик события?
34. Что такое колбэк (callback) функции в JavaScript?
35. Как работает асинхронный код в JavaScript, и какие механизмы используются для управления асинхронными операциями?

## 4 Фреймворки JavaScript

### 4.1 Введение в фреймворки JavaScript

#### 4.1.1 Понятие фреймворка и его роль в веб-разработке

Фреймворк JavaScript – это структура и набор инструментов, предназначенных для упрощения и ускорения процесса разработки веб-приложений. Фреймворк предоставляет готовые компоненты, библиотеки и структуру для построения приложений, что упрощает работу с веб-технологиями.

#### 4.1.2 Преимущества использования фреймворков перед чистым JavaScript

Использование фреймворков в веб-разработке имеет несколько значительных преимуществ:

- Ускорение разработки: Фреймворки предоставляют готовые решения, что уменьшает время, необходимое для создания приложения. Готовые компоненты и функции упрощают работу;
- Структурирование кода: Фреймворки внедряют архитектурные принципы, такие как MVC (Model-View-Controller) или MVVM (Model-View-ViewModel), что делает код более организованным и читаемым;
- Обширная поддержка и сообщество: Популярные фреймворки имеют большие сообщества разработчиков, что означает наличие множества ресурсов, документации и сторонних библиотек для поддержки.

#### 4.1.3 Ограничения и недостатки фреймворков

Несмотря на все преимущества, фреймворки также имеют свои ограничения и недостатки:

- Сложность: Изучение и работа с фреймворками может потребовать времени и усилий, особенно для новичков;
- Ограниченность гибкости: Фреймворки часто предоставляют определенную архитектуру и способы решения задач, что может ограничивать творчество разработчика;

- **Размер приложения:** Некоторые фреймворки имеют большой размер, что может влиять на скорость загрузки веб-приложения;
- **Обновления и совместимость:** не всегда обновления фреймворка совместимы с предыдущими версиями, что может потребовать дополнительной работы при обновлении приложения.

## **4.2 Основы TypeScript**

### **4.2.1 Введение в TypeScript**

TypeScript – это язык программирования, созданный на основе JavaScript, который предоставляет разработчикам инструменты для явного указания типов данных в коде, что делает его более надежным и удобным для разработки сложных приложений.

### **4.2.2 Типы данных и аннотации типов**

В TypeScript существует несколько встроенных типов данных, таких как:

- **number:** Числовой тип, представляющий как целые, так и вещественные числа;
- **string:** Строковый тип;
- **boolean:** Логический тип, который может быть true или false;
- **array:** Тип массива для хранения последовательности элементов одного типа;
- **object:** Объектный тип, представляющий собой структуру с определенными свойствами;
- **any:** Динамический тип, который позволяет присваивать переменным значения разных типов.

Аннотации типов позволяют указать тип переменной, параметра функции или возвращаемого значения функции, что помогает компилятору TypeScript проверить правильность использования типов в коде.

```

let age: number = 30;
let name: string = "John";
let isStudent: boolean = true;
let numbers: number[] = [1, 2, 3];
let person: { name: string, age: number } = { name: "Alice", age: 25 };

```

Рис. 4.1 - Пример создания переменных

### 4.2.3 Интерфейсы и классы в TypeScript

TypeScript поддерживает использование интерфейсов для определения структуры объектов. Это позволяет создавать более строго типизированный код и обеспечивает соответствие заданным интерфейсам.

```

interface Point {
  x: number;
  y: number;
}

1 usage
function printPoint(point: Point) : void {
  console.log(`x: ${point.x}, y: ${point.y}`);
}

const myPoint : {x: number, y: number} = { x: 10, y: 20 };
printPoint(myPoint);

```

Рис. 4.2 - Пример интерфейса

Кроме того, TypeScript позволяет создавать классы с явным указанием типов для свойств и методов, что облегчает разработку объектно-ориентированных приложений.

```

class Person {
  1 usage
  constructor(public name: string, public age: number) {}

  1 usage
  sayHello() : void {
    console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);
  }
}

const person : Person = new Person( name: "Alice", age: 30);
person.sayHello();

```

Рис. 4.3 - Пример класса

## 4.2.4 Преимущества TypeScript по сравнению с JavaScript

TypeScript предлагает несколько преимуществ по сравнению с чистым JavaScript:

- Статическая типизация: TypeScript позволяет выявлять и исправлять ошибки типов на этапе компиляции, что уменьшает вероятность ошибок во время выполнения;
- Улучшенная IDE-поддержка: многие современные интегрированные среды разработки (IDE) предоставляют более продвинутую поддержку для TypeScript, такую как автодополнение и быстрое обнаружение ошибок;
- Лучшая читаемость кода: Явное указание типов делает код более читаемым и понятным, что упрощает совместную работу и обслуживание кода;
- Более надежный код: благодаря статической типизации, TypeScript помогает предотвратить ошибки, связанные с типами данных, что способствует созданию более надежных приложений;
- Поддержка новых возможностей ECMAScript: TypeScript поддерживает новейшие возможности языка JavaScript и предоставляет средства для их использования в проектах.

## 4.3 Vue.js

### 4.3.1 Установка и настройка

Перед началом работы с Vue.js необходимо установить его и настроить окружение. Примеры реализации:

- Установка Vue.js с использованием npm или yarn:

```
npm install vue  
# или  
yarn add vue
```

- Создание нового проекта с помощью Vue CLI:

Vue CLI – это инструмент командной строки, который упрощает создание и управление проектами Vue.js. Требуется установить Vue CLI:

```
npm install -g @vue/cli  
# или  
yarn global add @vue/cli
```

Затем создайте новый проект с помощью команды:

```
vue create Имя проекта
```



- Знакомство с базовым проектом Vue и его структурой:

В созданном проекте вы увидите структуру, состоящую из файлов и папок, включая компоненты, шаблоны и конфигурационные файлы.

#### 4.3.2 Основы компонентной архитектуры

Vue.js основан на компонентной архитектуре, что позволяет создавать переиспользуемые компоненты.

Компоненты — это независимые блоки интерфейса, которые могут содержать шаблон, стили и логику.

```
<template>
  <button class="my-button">Click me</button>
</template>

<script>
no usages
export default {
  name: 'MyButton',
}
</script>

<style scoped>
.my-button {
  background-color: blue;
  color: white;
}
</style>
```

Рис. 4.4 - Пример Vue-компонента

Для того чтобы использовать компонент в другом компоненте или приложении, его необходимо зарегистрировать.

```

<template>
  <div>
    <my-button></my-button>
  </div>
</template>

<script>
import MyButton from './MyButton.vue';

no usages
export default {
  components: {
    'my-button': MyButton,
  }
}
</script>

```

Рис. 4.5 - Регистрация компонента

Пропсы позволяют передавать данные от родительского компонента к дочернему.

```

<!-- Родительский компонент -->
<template>
  <child-component message="Hello from parent"></child-component>
</template>

<!-- Дочерний компонент -->
<template>
  <div>{{ message }}</div>
</template>

<script>
no usages
export default {
  props: ['message'],
}
</script>

```

Рис. 4.6 - Пример props

Дочерний компонент может генерировать события и родительский компонент может на них реагировать:

```

<!-- Родительский компонент -->
<template>
  <child-component @custom-event="handleEvent"></child-component>
</template>

<script>
no usages
export default {
  methods: {
    handleEvent(data) {
      // Обработка события и данных
    }
  }
}
</script>

<!-- Дочерний компонент -->
<template>
  <button @click="emitEvent">Click me</button>
</template>

<script>
no usages
export default {
  methods: {
    emitEvent() {
      this.$emit( event: 'custom-event', args: 'Data to pass to parent');
    }
  }
}
</script>

```

Рис. 4.7 - Пример эвентов

### 4.3.3 Работа с директивами и шаблонами

Vue.js предоставляет директивы для управления DOM-элементами и данными в шаблонах:

- ☐ v-if - условное отображение элементов;

```

<template>
  <p v-if="showParagraph">This is a paragraph</p>
</template>

```

- ☐ v-for - повторение элементов для списка;

```

<template>
  <ul>
    <li v-for="item in items">{{ item }}</li>
  </ul>
</template>

```

- v-bind - привязка атрибутов;

```
<template>
  <a v-bind:href="url">Link</a>
</template>

<script>
no usages
export default {
  data() {
    return {
      url: 'https://example.com',
    };
  }
}
</script>
```

- v-on - обработка событий;

```
<template>
  <button v-on:click="handleClick">Click me</button>
</template>

<script>
no usages
export default {
  methods: {
    handleClick() {
      // Обработка клика
    }
  }
}
</script>
```

## 4.4 Задание для самоподготовки

### 4.4.1 Задача

В рамках этой лабораторной работы требуется разработать анонимный онлайн-чат, который позволит пользователям обмениваться сообщениями в реальном времени.

#### 4.4.2 Требования

- Выбор фреймворка: выбрать один из следующих фреймворков для разработки чата: Vue, React или Angular;
- подготовка окружения: создать пустой проект с выбранным фреймворком и настроить его для поддержки TypeScript;
- интерфейс чата: разработать пользовательский интерфейс для чата, включая окно чата, поле ввода сообщений и отображение имен пользователей;
- компоненты: разделить интерфейс на компоненты для лучшей организации кода;
- обмен сообщениями: реализовать функциональность обмена сообщениями в реальном времени между пользователями с использованием WebSocket. Сообщения должны быть в формате JSON и содержать следующие поля: name (имя пользователя) и message (текст сообщения);
- аутентификация: включить возможность, чтобы пользователи могли указать свое имя перед началом чата (простая аутентификация).
- бэкенд WebSocket сервер: разработать простой бэкенд на Node.js для создания WebSocket сервера, обеспечивающего обмен сообщениями между клиентами.

#### 4.4.3 Варианты

1. Реализовать возможность отправки изображений: добавить функциональность для загрузки и отображения изображений в чате, позволяя пользователям обмениваться фотографиями.
2. Создать режим анонимности: предоставить пользователям возможность войти в режим анонимности, скрывая свое имя от других пользователей.
3. Перевести интерфейс на другие языки: добавить мультиязычную поддержку, позволяя пользователям выбирать язык интерфейса.
4. Добавить поддержку многопользовательских чатов: расширить функциональность для создания и участия в групповых чатах с несколькими пользователями.

5. Реализовать эмодзи: добавить поддержку эмодзи для более выразительного общения.
6. Реализовать реакции на сообщения: добавить возможность при наведении на сообщение выбрать один из нескольких предложенных эмодзи.

### **Список литературы**

1. [Online] // ECMAScript. - <https://262.ecma-international.org/>.