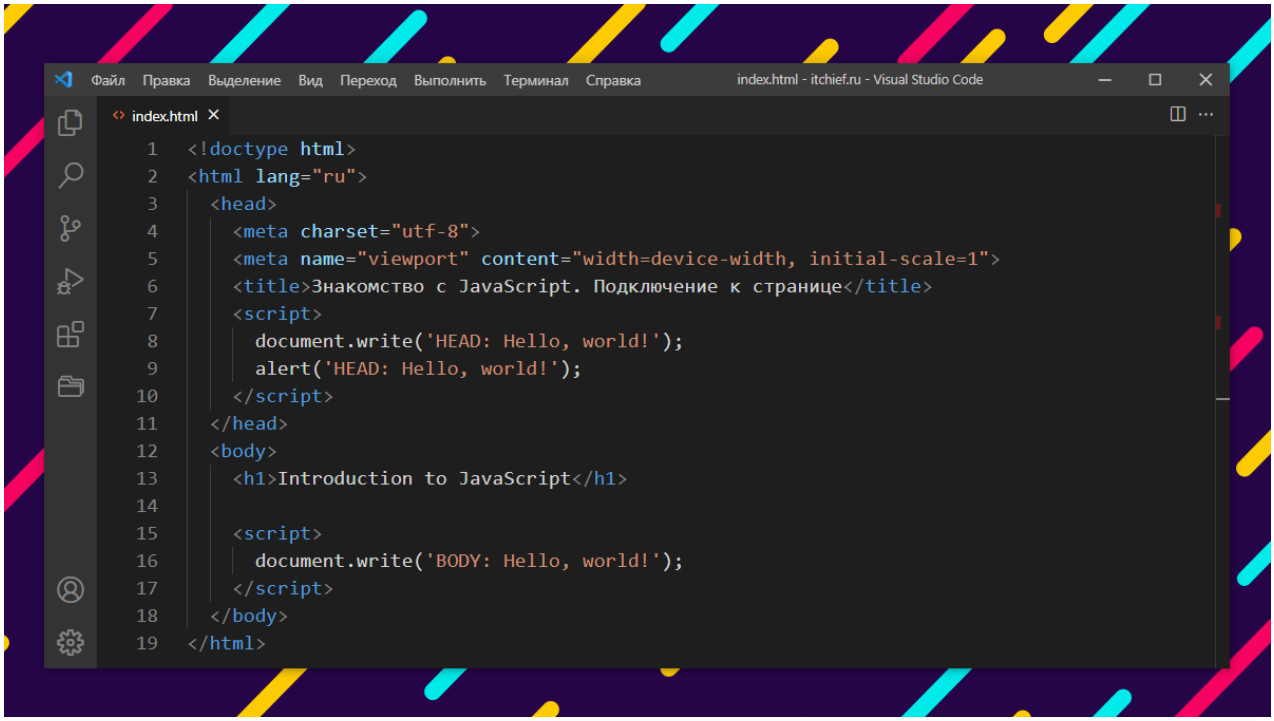
Лабораторная работа №1. Знакомство с JavaScript.

Подключение к странице



Содержание:

На этом занятии разберем, что такое JavaScript и для чего он нужен. После этого рассмотрим различные варианты подключения кода JavaScript к странице.

Что такое JavaScript?

JavaScript (сокращённо JS) – это **язык программирования**, который изначально был придуман для браузера, чтобы придать страницам **интерактивность и динамичность**.

Программы написанные на JavaScript называются **сценариями** или **скриптами**. Добавление их на страницу выполняется через тег `<script>`. При этом их можно как непосредственно вставлять на веб-страницу, так и размещать в отдельном файле. Каждый браузер имеет интерпретатор JavaScript, с помощью которого он выполняет этот код.

Потребность в создании языка программирования для веб-браузера возникла в 90-е годы. В это время на веб-страницах хотелось делать уже намного больше, чем просто выводить статичный контент.

Данный язык изначально был создан Бренданом Айком в 1995 году для браузера Netscape и назывался он Mocha. Но после ряда переименований он обрёл окончательное имя – **JavaScript**. В начале он имел очень маленькую функциональность и в основном использовался для добавления на страницу интерактивности. Но со временем язык стал развиваться и позволять делать всё больше и больше.

В 1996 году язык JavaScript был стандартизован компанией Ecma, которая занимается стандартизацией информационных и коммуникационных технологий. Сама спецификация была названа ECMAScript или сокращённо ES. По сути, JavaScript является реализацией спецификации ECMAScript. Новые версии ECMAScript выходят ежегодно и добавляют в язык новые возможности.

Версии:

- ECMAScript 1 – июнь 1997;
- ECMAScript 2 – июнь 1998;
- ECMAScript 3 – декабрь 1999;
- ECMAScript 5 – декабрь 2009;
- ECMAScript 5.1 – июнь 2011;
- ECMAScript 2015 (ES2015, ECMAScript 6, ES6) – июнь 2015;
- ECMAScript 2016 (ES2016, ECMAScript 7, ES7) – июнь 2016;
- ECMAScript 2017 (ES2017, ECMAScript 8, ES8) – июнь 2017;
- ECMAScript 2018 (ES2018, ECMAScript 9, ES9) – июнь 2018;
- ECMAScript 2019 (ES2019, ECMAScript 10, ES10) – июнь 2019;
- ECMAScript 2020 (ES2020, ECMAScript 11, ES11) – июнь 2020;
- ECMAScript 2021 (ES2021, ECMAScript 12, ES12) – июнь 2021.

ECMAScript – это стандарт JavaScript, который описывает полностью все функции JavaScript. Браузеры, Node.js применяют это стандарт и реализуют его поддержку.

Начиная с версии ES6 язык значительно преобразился, в нём появился новый синтаксис для написания сложных приложений (классы, модули, итераторы, циклы, генераторы в стиле Python, стрелочные функции, ключевые слова `let`, `const` и многое другое). С выходом версии ES6 весь код JavaScript принято делить на **modern (современный)** и **классический (до ES6)**.

В настоящее время **JavaScript применяется не только в веб-браузере**. С помощью него можно писать десктопные и мобильные приложения, использовать его на сервере (Node.js).

Язык JavaScript, как и другие языки программирования, имеет некоторые свои особенности. Среди основных – **слабая типизация** и **динамическое приведение типов**.

JavaScript – это **не Java**, хоть он и унаследовал некоторые синтаксические конструкции этого языка. Такое название данный язык получил в силу некоторых исторических причин. Одной из них являлось то, что изначально в качестве языка, который должен был быть доступным в браузере, хотели сделать **Java**. Но впоследствии компания **Netscape** отказалась от этой мысли, из-за того, что **Java** был слишком большим и сложным.

Виды браузеров и браузерных движков

В настоящее время существует большое количество **браузеров**. Любой современный браузер основывается на **движке**. **Движок** – это часть браузера, которая преобразует HTML, CSS, JavaScript, изображения и другую информацию в **интерактивную картинку**.

Основные современные движки и браузеры, которые их используют:

- **Blink** (Google Chrome, Opera, Яндекс.Браузер и др.);
- **Gecko** (Mozilla Firefox, Waterfox и др);
- **WebKit** (Safari, Maxthon, Vivaldi и др.);
- **EdgeHTML** (Microsoft Edge).

Подключение JavaScript к странице

Добавление JavaScript на страницу выполняется с помощью тега `<script>`.

Первый способ – это вставка кода **непосредственно на страницу**. Выполняется это между открывающим и закрывающим тегом `<script>`.

JavaScript Копировать
```
<script>
  alert('Привет, мир!');
</script>
```
Второй способ заключается в **использовании отдельного файла с расширением js**. В данный файл необходимо поместить код JavaScript, а затем подключить его к странице с помощью `<script>`. Путь к файлу задаётся с помощью атрибута `src`.

JavaScript Копировать
```
<script src="main.js"></script>
```
С помощью этого способа можно подключить JavaScript к большому количеству HTML страниц. Это позволяет при изменении кода не править его на каждой странице.

`<script>` можно поместить внутрь любого элемента, но рекомендуется непосредственно в `<head>` или `<body>`:

JavaScriptКопировать
```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Знакомство с JavaScript. Подключение к странице</title>
    <!-- Скрипт в <head> -->
    <script src="main-head.js"></script>
  </head>
  <body>
    ...

    <!-- Скрипт перед закрывающим тегом <body> -->
    <script src="main.js"></script>
  </body>
</html>
```

Если подключить скрипт с помощью атрибута `src` и дополнительно ещё указать **некоторый код** между открывающим и закрывающим тегом `script`, то код, который вы указали непосредственно, будет проигнорирован, т.е. он не выполнится.

JavaScriptКопировать
```
<!-- Будет выполнен только скрипт common.js -->
<script src="common.js">
  alert('Это предупреждение никогда не отобразится!');
</script>
```

Как выполняются скрипты на странице?

Когда браузер читает страницу и встречает на ней `script`, он **останавливает дальнейшую загрузку страницы** и выполняет, подключенный с помощью этого элемента JavaScript. После чего приступает к дальнейшей загрузке страницы:

JavaScriptКопировать
```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Знакомство с JavaScript. Подключение к странице</title>
    <!-- Скрипт в <head> -->
    <script>
      document.write('HEAD: Hello, world!');
      alert('HEAD: Hello, world!');
    </script>
  </head>
  <body>
    <h1>Introduction to JavaScript</h1>

    <script>
      document.write('BODY: Hello, world!');
```

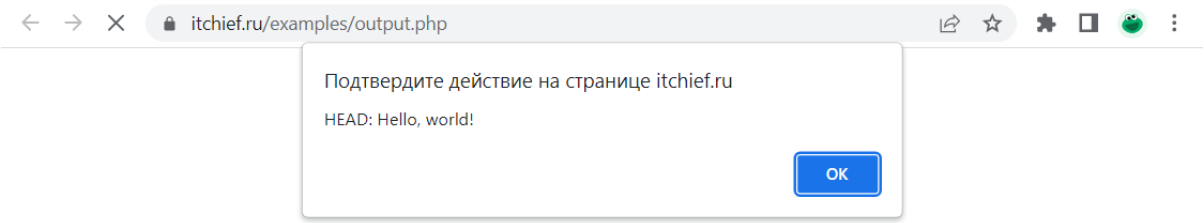
```
    </script>
  </body>
</html>
```

В этом примере используются 2 метода:

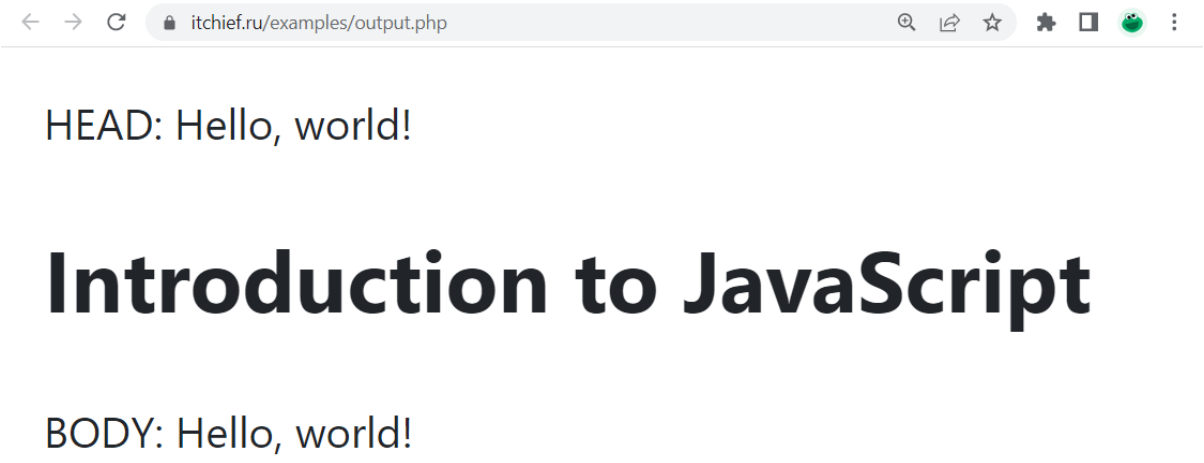
- `alert()` – для вывода на экран предупреждения с кнопкой «OK»; данное предупреждение приостанавливает дальнейшую загрузку страницу;
- `document.write()` – для вывода текста в текущее место HTML, если страница ещё не загрузилась.

При загрузке этого документа, когда браузер встретит `alert()`, он выведет на экран предупреждение «HEAD: Hello, world!». Кроме этого, он **приостановит дальнейшую загрузку страницы**, пока пользователь не нажмёт на кнопку «OK».

На этом этапе загрузки страницы:



После нажатия на кнопку «OK» браузер продолжит загрузку страницы. Он выполнит метод `document.write()`, который выведет значение, указанное в скобках сразу после открывающего тега `<body>`. Затем тег `<h1>` и в завершении выполнит метод `document.write()`, который выведет текст «BODY: Hello, world!».



async и defer

До появления этих атрибутов у нас не было возможности загружать JavaScript в фоне. Поэтому многие сайты, в которых JavaScript не использовался для

формирования первоначальной структуры страницы, подключали его в самом конце, т.е. перед закрывающим тегом `<body>`. В основном это использовалось для того, чтобы пользователь мог как можно быстрее увидеть страницу и начать с ней взаимодействовать. А так как такой код взаимодействует с уже загруженной страницей, его в принципе и вставлять выше не имеет смысла.

`async` – атрибут для `<script>`, который используется для того, чтобы указать браузеру на то, что этот скрипт необходимо загрузить асинхронно, т.е. не останавливая основной поток чтения страницы. После загрузка скрипта, браузер сразу же его выполнит.

JavaScript Копировать
```
<script src="/path/to/app.js" async></script>
```
`defer` – это ещё один атрибут для `<script>`, похожий на `async`. Он также указывает, что скрипт необходимо загрузить в фоне. Но в отличие от `async`, он будет выполнен после загрузки страницы, а точнее DOM. Кроме этого, стоит отметить, что скрипты, заданные с атрибутом `async`, выполняются перед вызовом события `DOMContentLoaded`.

JavaScript Копировать
```
<script src="/path/to/app.js" defer></script>
```
Ещё одно отличие `defer` от `async` в том, что `defer` сохраняет очередность их выполнения:

JavaScript Копировать
```
<script src="/path/to/script-1.js" defer></script>
<script src="/path/to/script-2.js" defer></script>
```
Т.е. вне зависимости от того какой скрипт загрузится быстрее, они всё равно будут выполнены браузером в том порядке, в котором они расположены в коде. В данном примере, сначала выполнится «script-1.js», а затем «script-2.js» даже если второй загрузится быстрее, чем первый.

Если для `<script>` одновременно указать два атрибута, т.е. сразу `async` и `defer`, то будет работать только `async`.

JavaScript Копировать
```
<script src="path_to/script.js" async defer></script>
```
Атрибуты `async` и `defer` можно использовать только для скриптов, подключаемых на страницу с использованием `src`.