Методичка по темам: «Основы HTML,CSS.Основы JavaScript.Стандарты»

Авторы: Бжезинская Евгения, Пивоварчик Виктория, Дубатовка Елизавета, Шинкевич Александр, Новиков Кирилл.

Доска проекта: <[https://trello.com/b/FpYPeJnv/sss](https://vk.com/away.php?to=https%3A%2F%2Ftrello.com%2Fb%2FFpYPeJnv%2Fsss&cc_key=)>

Оглавление:

[**ГЛАВА 1.** **ОСНОВЫ HTML.** 3](#_Toc21384107)

[**1.1 ОСНОВНЫЕ ПОНЯТИЯ HTML** 3](#_Toc21384108)

[**ГЛАВА 2. ОСНОВЫ CSS** 19](#_Toc21384109)

[**2.1 ОПРЕДЕЛЕНИЕ CSS** 19](#_Toc21384110)

[**2.2 ОБЩИЙ СИНТАКСИС ТАБЛИЦ СТИЛЕЙ** 21](#_Toc21384111)

[**2.3 ПРАВИЛА CSS** 22](#_Toc21384112)

[**2.4 КЛАССЫ** 24](#_Toc21384113)

[**2.5 ВСТРОЕННЫЙ СТИЛИ** 26](#_Toc21384114)

[**2.6 ВНЕДРЕННЫЕ СТИЛИ** 27](#_Toc21384115)

[**2.7 СВЯЗАННЫЕ ТАБЛИЦЫ СТИЛЕЙ** 28](#_Toc21384116)

[**2.8 АППАРАТНО-ЗАВИСИМЫЕ СТИЛИ** 29](#_Toc21384117)

[**2.9 ИДЕНТИФИКАТОРЫ** 30](#_Toc21384118)

[**2.10 СВОЙСТВА CSS** 31](#_Toc21384119)

[**ГЛАВА 3. СКРИПТЫ. JAVA SCRIPT** 35](#_Toc21384120)

[**3.1 ЧТО ТАКОЕ СКРИПТЫ И ИХ ВИДЫ** 35](#_Toc21384121)

[**3.2 ОСНОВНЫЕ ПОНЯТИЯ JAVA SCRIPT** 39](#_Toc21384122)

[**3.3 ВОЗМОЖНОСТИ JS** 47](#_Toc21384123)

[**3.4 РЕДАКТОРЫ КОДА** 51](#_Toc21384124)

[**3.5 КНИГИ, СПРАВОЧНИКИ И КЛАССИФИКАЦИИ** 53](#_Toc21384125)

[**ГЛАВА 4. СТАНДАРТЫ** 55](#_Toc21384126)

[**4.1 КОМПАНИЯ, УТВЕРЖДАЮЩАЯ СТАНДАРТЫ HTML** 55](#_Toc21384127)

[**4.2 РОЛЬ В РАЗРАБОТКЕ СТАНДАРТОВ И ВЗАИМОДЕЙСТВИЕ С W3C** 56](#_Toc21384128)

[**4.3 ВЕРСИИ** 57](#_Toc21384129)

[**4.4 КОДИРОВКА** 59](#_Toc21384130)

[**ВОПРОСЫ ПО ИЗУЧЕННОМУ МАТЕРИАЛУ:** 60](#_Toc21384131)

[**ИСТОЧНИКИ:** 62](#_Toc21384132)

## **ГЛАВА 1.** **ОСНОВЫ HTML.**

## **ОСНОВНЫЕ ПОНЯТИЯ HTML**

HTML- это язык разметки гипертекста (от англ. *HyperText Markup Language*).

Данный язык применяется для создания веб-страниц. Он интерпретируется (обрабатывается) браузером и отображается в виде документа в удобной для человека форме.

HTML – это неотъемлемая составляющая и основа практически любой веб-страницы. Язык HTML в первую очередь выступает как средство логической разметки страницы.

Именно HTML позволяет нам наделять содержимое страницы определенным смыслом, а реализуется это с помощью так называемых тэгов.

Тэги – это специальные маркеры, которые определенным образом интерпретируются браузером. Суть тэгов в том, что содержимое страницы, заключенное в разные тэги, по-разному обрабатывается браузером.

Скажем, мы можем заключить контент (содержимое) страницы в тэг параграфа, и данное содержимое будет считаться браузером параграфом.

Мы можем заключить содержимое в тэг списка, и тогда информация внутри него будет интерпретироваться как список.

Мы можем поместить контент в тэг таблицы, и содержимое в итоговом документе будет отображаться в виде таблицы и т.д.

Иными словами, каждый из HTML-тэгов, предназначенных для разметки текстовой информации, *придает этой информации некоторый смысл*.

Язык HTML имеет достаточно длительную историю развития, и за это время претерпел значительные изменения. Большая часть изменений связана с добавлением в язык новых тэгов и «выводом из обращения» устаревших. На момент написания этой статьи уже была начата разработка версии HTML 5.3.

Однако такой подход считается неверным и устаревшим, т.к. для задания внешнего вида существует отдельный язык.

На этом с HTML мы завершаем и переходим к языку CSS.

Расширение файлов: *.css*

CSS – это язык описания внешнего вида документа, написанного с использованием языка разметки. Название произошло от англ. *Cascading Style Sheets* — каскадные таблицы стилей.

Проще говоря, язык CSS предназначен для того, чтобы придавать необходимый внешний вид HTML-документам.

Придание внешнего вида документам HTML – это хоть и самый популярный, однако лишь частный случай применения языка CSS, т.к. с его помощью можно придавать вид и документам других типов: *XHTML, SVG* и *XUL*. Про них мы отдельно говорить не будем, т.к. это выходит за рамки рассматриваемого вопроса.

Итак, целью создания CSS было отделение описания логической структуры веб-страницы от ее внешнего вида. Как вы уже знаете, для описания структуры используется HTML, для описания же того, как эта логическая структура будет выглядеть, отвечает как раз CSS.

Раздельное описание логической структуры и представления документа позволяет более гибко управлять внешним видом документа и минимизировать объем повторяющегося кода, который бы неизбежно возникал при использовании HTML для описания внешнего вида документа.

С помощью CSS веб-разработчик может задавать для страницы и отдельных ее элементов различные гарнитуры и размеры шрифта, цвета элементов, отступы элементов друг от друга, расположение отдельных блоков на странице и т.д.

Разумеется, для того, чтобы использовать CSS для придания внешнего вида HTML-документу, нужно этот документ как-то связать со стилями, т.е. «сообщить» HTML-документу, что он будет оформлен с помощью CSS.

Для этого существуют различные способы подключения CSS к документу, которые дают браузеру знать, что к странице в целом, либо к каким-то отдельным ее элементам должно быть применено стилевое оформление.

Таблицы стилей могут располагаться как непосредственно внутри того, документа, к которым они будут применяться, так и находиться в отдельном файле, имеющем расширение *.css*.

Важно понимать, что *CSS-файл – это обычный текстовый файл*. В нем пишутся специальные инструкции, описывающие внешний вид элемента и его позиционирование на странице а также комментарии (произвольные пояснения относительно написанных инструкций).

Стили CSS могут быть подключены к странице (либо внедрены в нее) четырьмя следующими способами:

*1.* Если таблица стилей находится в отдельном файле, то она подключается к документу с помощью специального тэга *link*, который должен располагаться в этом документе внутри тэга *head*:

<head>

<link rel="stylesheet" type="text/css" href="style.css">

</head>

В атрибуте *href* при этом указывается путь до подключаемого файла стилей.

*2.* Второй способ подключения применяется также в том случае, если стили находятся в отдельном файле. При этом используется директива *@import*, которая должна находиться в этом документе внутри тэгов *style* (которые, в свою очередь, должны находиться внутри тэгов *head*):

<head>

<style type="text/css" media="all">

@import url(style.css);

</style>

</head>

Здесь, как вы можете видеть, путь до подключаемого файла указывается в круглых скобках после слова *url*.

*3.* Третий способ используется в случае, когда стили CSS расположены внутри того документа, к которому они должны применяться. В этом случае стили должны находиться в этом документе внутри тэгов *style* (которые, в свою очередь, должны находиться внутри тэгов *head*):

<head>

<style type="text/css">

body {

color: red;

}

</style>

</head>

В примере выше внутри тэгов *style* расположен стиль, задающий красный цвет текста для элемента *body* веб-страницы.

*4.* Четвертый способ также применяется в случаях, когда таблица стилей описана в самом документе. При этом стилевое оформление задается для какого-то конкретного элемента веб-страницы (тэга) посредством использования его атрибута *style*.

При этом все CSS-правила, заданные таким образом, будут применяться только к текущему тэгу (элементу веб-страницы).

<p style="font-size: 21px; color: green;">Используем встроенные таблицы стилей.</p>

В примере выше мы задаем размер шрифта и его цвет для одного конкретного абзаца текста в рамках документа.

Подводя итог по способам подключения стилей, можно сказать, что в первых двух случаях к документу применялись внешние таблицы стилей, в то время как в 3 и 4 способах мы имеем дело со встроенными (или внутренним) стилями.

Сами стилевые правила задаются в виде пар *«свойство: значение;»*. При этом применяются эти правила к так называемым селекторам. Скажем, в 3 примере мы имеем дело с селектором элемента (*body*). В качестве свойства выступает *color*, а в качестве значения – *red*.

Всего существует 10 видов селекторов, рассмотрение которых, по понятным причинам, выходит за рамки данного обзора.

Важно, чтобы вы понимали общий принцип.

*Селектор определяет один или группу элементов, к которым будут применяться стилевые правила* (в нашем случае это все содержимое тэга *body*)

Свойство можно определить как вид преобразования, которое будет применено к элементу (в нашем случае это *color*, т.е. цвет текста).

Значение же определяет непосредственно значение определенного вида преобразования (в нашем примере это *red*, т.е. красный цвет).

В процессе своего развития язык CSS уже прошел достаточно длинный путь, и в настоящее время существует несколько его уровней: CSS1, CSS2, CSS2.1, CSS3. C конца 2011 года разрабатывается уже CSS4.

Смысл разных уровней в том, что происходит исправление существующих ошибок, добавление новых свойств, расширение механизма селекторов и т.д.

Иными словами, каждый следующий уровень является не чем-то обособленным, а логическим развитием и продолжением предыдущего уровня, позволяя более тонко и гибко управлять внешним видом web-страниц.

* 1. **HTML**

Структура HTML страницы

HTML является стандартным языком разметки для создания веб-страниц.

HTML расшифровывается язык гипертекстовой разметки

HTML описывает структуру веб-страницы

HTML состоит из ряда элементов

HTML элементы сообщить браузеру, как отображать содержимое

HTML элементы представлены теги

HTML-теги этикетки куски контента, таких как «заголовок», «пункт», «стол», и так далее

Браузеры не отображают HTML-тег, но использовать их для отображения содержимого страницы

Простой HTML документ

пример

<!DOCTYPE html>  
<html>  
<head>  
<title>Page Title</title>  
</head>  
<body>  
  
<h1>My First Heading</h1>  
<p>My first paragraph.</p>  
  
</body>  
</html>

Пример Разъяснения

<!DOCTYPE html>Декларация определяет этот документ , чтобы быть HTML5

<html>Элемент является корневым элементом HTML страницы

<head>Элемент содержит мета - информацию о документе

<title>Элемент определяет заголовок для документа

<body>Элемент содержит видимое содержимое страницы

<h1>Элемент определяет большой заголовок

<p>Элемент определяет пункт

**HTML теги**

HTML-теги имена элементов в окружении угловых скобок:

<Тэг> содержание идет здесь ... </ тэг>

HTML - теги обычно приходят парами , как <p>и</p>

Первый тег в паре является начальным тегом, второй тег является конечным тегом

Конечный тег записывается как начальный тег, но с косой чертой , вставленной перед именем тега

Совет: Открывающий тег также называется открывающий тег , а конец тег закрывающий тег .

веб-браузеры

Цель веб-браузера (Chrome, пограничный, Firefox, Safari), чтобы прочитать HTML документы и отображать их.

Браузер не отображает HTML тегов, но использует их, чтобы определить, как отображать документ:

рис. 1

Структура HTML страницы

Ниже визуализация структуры HTML-страницы:

<HTML>

<Голова>

<Название> Заголовок страницы </ title>

</ HEAD>

<Тело>

<H1> Это заголовок </ h1>

<Р> Это параграф. </ P>

<Р> Это еще один параграф. </ P>

</ Body>

</ Html>

Примечание: Только содержимое внутри <тела> секции (белая область выше) отображается в браузере.

<! DOCTYPE> Декларация

<!DOCTYPE>Декларация представляет собой тип документа, а также помогает браузеры корректно отображать веб - страницы.

Он должен появиться только один раз, в верхней части страницы (перед любым HTML-теги).

<!DOCTYPE>Декларация не чувствительна к регистру.

<!DOCTYPE>Декларация HTML5 является:

<!DOCTYPE html>

Написать HTML Использование блокнота или TextEdit

Веб-страницы могут быть созданы и изменены с помощью профессиональных HTML-редакторов.

Однако, для изучения HTML мы рекомендуем простой текстовый редактор, например Блокнот (PC) или TextEdit (Mac).

Мы считаем, что с помощью простого текстового редактора, это хороший способ узнать HTML.

Выполните следующие действия, чтобы создать свой первый веб-страницу с помощью блокнота или TextEdit.

Шаг 1: Откройте Блокнот (ПК)

**Windows 8 или более поздняя версия:**

Откройте **экран Start** (символ окна в нижнем левом углу на экране). Введите**Блокнот** .

**Windows 7 или более ранних версий:**

Откройте **Пуск** > **Программы> Стандартные> Блокнот**

Шаг 1: Откройте TextEdit (Mac)

Откройте **Finder> Приложения> TextEdit**

Также изменить некоторые предпочтения, чтобы получить приложение, чтобы правильно сохранять файлы. В **Настройках>Формат>**выберите **«Обычный текст»**

Затем в разделе «Открыть и Сохранить», установите флажок с надписью «Показать HTML файлы как HTML код вместо форматированный текст».

**Затем откройте новый документ, чтобы разместить код.**

Шаг 2: Написать Некоторый HTML

Написать или скопировать некоторые HTML в Блокнот.

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Heading</h1>  
  
<p>My first paragraph.</p>  
  
</body>  
</html>

 рис. 2

Шаг 3: Сохраните HTML-страницу

Сохраните файл на вашем компьютере. Выберите **Файл> Сохранить как** в меню Блокнота.

Назовите файл **«index.htm»** и установить кодировку **UTF-8** (который является предпочтительным для кодирования HTML - файлов).

рис. 3

Вы можете использовать либо .htm или .html в качестве расширения файла. Там нет никакой разницы, это до вас.

Шаг 4: Просмотр HTML-страницы в браузере

Откройте сохраненный HTML-файл в вашем любимом браузере (двойной щелчок на файле, или щелкните правой кнопкой мыши - и выберите «Открыть с помощью»).

Результат будет выглядеть так же, как это:

рис. 4

**HTML документы**

Все HTML - документы должны начинаться с объявления типа документа: <!DOCTYPE html>.

Сам HTML-документ начинается с <html>и заканчивается на </html>.

Видимая часть документа HTML находится между <body>и </body>.

пример

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Heading</h1>  
<p>My first paragraph.</p>  
  
</body>  
</html>

**HTML-заголовки**

Заголовки HTML определяются тегами <h1>to <h6>.

<h1>определяет наиболее важный заголовок. <h6>определяет наименее важный заголовок:

пример

<h1>This is heading 1</h1>  
<h2>This is heading 2</h2>  
<h3>This is heading 3</h3>

**HTML параграфы**

HTML-абзацы определяются с помощью <p>тега:

пример

<p>This is a paragraph.</p>  
<p>This is another paragraph.</p>

**HTML-ссылки**

HTML-ссылки определяются с помощью <a>тега:

пример

<a href="https://www.w3schools.com">This is a link</a>

Назначение ссылки указывается в hrefатрибуте.

Атрибуты используются для предоставления дополнительной информации об элементах HTML.

Вы узнаете больше об атрибутах в следующей главе.

**HTML изображения**

HTML-изображения определяются с помощью <img>тега.

Исходный файл ( src), альтернативный текст ( alt) widthи height предоставляются в виде атрибутов:

пример

<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">

HTML-кнопки

HTML-кнопки определяются с помощью <button>тега:

пример

<button>Click me</button>

**HTML списки**

Списки HTML определяются с помощью тега <ul>(неупорядоченный / маркированный список) или <ol> (упорядоченный / нумерованный список), за которым следуют <li> теги (элементы списка):

пример

<ul>  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ul>  
  
<ol>  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ol>

**HTML атрибуты**

Все элементы HTML могут иметь атрибуты

Атрибуты предоставляют дополнительную информацию об элементе

Атрибуты всегда указываются в начальном теге

Атрибуты обычно входят в пары имя / значение, такие как: name = "value"

**Атрибут href**

HTML-ссылки определяются с помощью <a>тега. Адрес ссылки указан в hrefатрибуте:

пример

<a href="https://www.w3schools.com">This is a link</a>

Атрибут src

HTML-изображения определяются с помощью <img>тега.

Имя файла источника изображения указывается в srcатрибуте:

пример

<img src="img\_girl.jpg">

**Атрибуты ширины и высоты**

 HTML-изображения также имеют width и height атрибуты, которые определяют ширину и высоту изображения:

пример

<img src="img\_girl.jpg" width="500" height="600">

Ширина и высота указываются в пикселях по умолчанию; ширина = 500 означает ширину 500 пикселей.

**Атрибут alt**

altАтрибут определяет альтернативный текст , который будет использоваться, если изображение не может быть отображено.

Значение altатрибута может быть прочитано программами чтения с экрана. Таким образом, кто-то, «слушающий» веб-страницу, например, человек с нарушениями зрения, может «услышать» этот элемент.

пример

<img src="img\_girl.jpg" alt="Girl with a jacket">

altАтрибут также полезно , если изображение не может быть отображено (например , если он не существует):

пример

Посмотрим, что произойдет, если мы попытаемся отобразить несуществующее изображение:

<img src="img\_typo.jpg" alt="Girl with a jacket">

**Атрибут стиля**

Style Атрибут используется для указания стиля элемента, как цвет, шрифт, размер и т.д.

пример

<p style="color:red">This is a paragraph.</p>

**Атрибут lang**

Язык документа может быть объявлен в <html>теге.

Язык объявлен с lang атрибутом.

Объявление языка важно для приложений с расширенными возможностями (средства чтения с экрана) и поисковых систем:

<!DOCTYPE html>  
<html lang="en-US">  
<body>  
  
...  
  
</body>  
</html>

Первые две буквы указывают на язык (en). Если есть диалект, добавьте еще две буквы (США).

**Атрибут заголовка**

Здесь titleатрибут добавляется к <p> элементу. Значение атрибута title будет отображаться в виде всплывающей подсказки при наведении указателя мыши на абзац:

пример

<p title="I'm a tooltip">  
This is a paragraph.  
</p>

**HTML-теги комментариев**

Вы можете добавлять комментарии к своему источнику HTML, используя следующий синтаксис:

<!-- Write your comments here -->

Обратите внимание, что в открывающем теге есть восклицательный знак (!), А в закрывающем - нет.

**Примечание.** Комментарии не отображаются в браузере, но могут помочь документировать исходный код HTML.

С комментариями вы можете размещать уведомления и напоминания в своем HTML:

пример

<!-- This is a comment -->  
  
<p>This is a paragraph.</p>  
  
<!-- Remember to add more information here -->

Комментарии также отлично подходят для отладки HTML, потому что вы можете закомментировать строки кода HTML, по одной за раз, для поиска ошибок:

пример

<!-- Do not display this image at the moment  
<img border="0" src="pic\_trulli.jpg" alt="Trulli">  
-->

## **ГЛАВА 2. ОСНОВЫ CSS**

## **2.1 ОПРЕДЕЛЕНИЕ CSS**

**CSS** – это язык описания внешнего вида документа, написанного с использованием языка разметки. Название произошло от англ. *Cascading Style Sheets* — каскадные таблицы стилей.

Проще говоря, язык CSS предназначен для того, чтобы придавать необходимый внешний вид HTML-документам.

Придание внешнего вида документам HTML – это хоть и самый популярный, однако лишь частный случай применения языка CSS, т.к. с его помощью можно придавать вид и документам других типов: *XHTML, SVG* и *XUL*. Про них мы отдельно говорить не будем, т.к. это выходит за рамки рассматриваемого вопроса.

Итак, целью создания CSS было **отделение описания логической структуры веб-страницы от ее внешнего вида**. Как вы уже знаете, для описания структуры используется HTML, для описания же того, как эта логическая структура будет выглядеть, отвечает как раз CSS.

Раздельное описание логической структуры и представления документа позволяет более **гибко управлять внешним видом документа и минимизировать объем повторяющегося кода**, который бы неизбежно возникал при использовании HTML для описания внешнего вида документа.

С помощью CSS веб-разработчик может задавать для страницы и отдельных ее элементов различные гарнитуры и размеры шрифта, цвета элементов, отступы элементов друг от друга, расположение отдельных блоков на странице и т.д.

Разумеется, для того, чтобы использовать CSS для придания внешнего вида HTML-документу, нужно этот документ как-то связать со стилями, т.е. «сообщить» HTML-документу, что он будет оформлен с помощью CSS.

Для этого существуют различные способы подключения CSS к документу, которые дают браузеру знать, что к странице в целом, либо к каким-то отдельным ее элементам должно быть применено стилевое оформление.

Таблицы стилей могут располагаться как непосредственно внутри того, документа, к которым они будут применяться, так и находиться в отдельном файле, имеющем расширение *.css*.

Важно понимать, что *CSS-файл – это обычный текстовый файл*. В нем пишутся специальные инструкции, описывающие внешний вид элемента и его позиционирование на странице а также комментарии (произвольные пояснения относительно написанных инструкций).

**2.2 ОБЩИЙ СИНТАКСИС ТАБЛИЦ СТИЛЕЙ**

Таблицы стилей строятся в соответствии с определенным порядком (синтаксисом), в противном случае они не могут нормально работать. Таблицы стилей составляются из определенных частей (рис. 1):

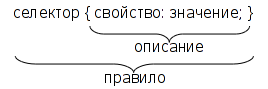


Рис. 1. Синтаксис описания стиля CSS

* **Селектор (Selector).** Селектор — это элемент, к которому будут применяться назначаемые стили. Это может быть тег, класс или идентификатор объекта гипертекстового документа.
* **Свойство (Property).** Свойство определяет одну или несколько характеристик селектора. Свойства задают формат отображения селектора: отступы, шрифты, выравнивание, размеры и т.д.
* **Значение (Value).** Значения — это фактические числовые или строковые константы, определяющие свойство селектора.
* **Описание (Declaration)**. Совокупность свойств и их значений.
* **Правило (Rule).** Полное описание стиля (селектор + описание).

Таким образом, таблица стилей — это набор правил, задающих значения свойств селекторов, перечисленных в этой таблице. Общий синтаксис описания правила выглядит так:

селектор[, селектор[, ...]] {свойство: значение;}

Регистр символов значения не имеет, порядок перечисления селекторов в таблице и свойств в определении не регламентирова

**2.3 ПРАВИЛА CSS**

Итак, **каскадная таблица стилей** — это набор правил форматирования тегов HTML. Приведем несколько примеров написания таких правил:

1. Основной текст с выравниванием по ширине, абзацный отступ 30px, гарнитура (шрифт) — Serif, кегль (размер шрифта) — 14px:

p {

text-align: justify;

text-indent: 30px;

font-family: Serif;

font-size: 14px;

}

Это правило будет применено ко всем тегам <p>.

1. Синий цвет для заголовков с первого по третий уровень:

h1, h2, h3 {

color: blue; /\* тоже самое, что и *#0000FF \*/*

}

1. Таблицы и изображения выводить без обрамления:

table, img {border: none;}

1. Ссылки в элементах списков показывать без подчеркивания:

li a {text-decoration: none;}

1. Внутренние отступы слева и справа для блоков (<div>), заголовков таблиц и ячеек таблиц установить в 10px и залить фон желтым цветом:

div, th, td {

padding-left: 10px;

padding-right: 10px;

background-color: yellow;

}

1. Все ссылки в документе отображать черным цветом и полужирным шрифтом, а в основном тексте и списках — обычным, а также выделять их зеленым цветом и подчеркивать только при наведении курсора (в описании правил использован псевдоэлемент a:hover).

a {color: black; font-weight: bold;}

p a, li a {font-weight: normal; text-decoration: none;}

p a:hover, li a:hover {

color: *#00FF00; text-decoration: underline;*

}

**2.4 КЛАССЫ**

Стандарт CSS представляет возможности создания именованных стилей — стилевых классов. Это позволяет ответить на такой, например, вопрос: Как применить разные стили к одному и тому же селектору?

Предположим, что в документе вам нужны два различных вида основного текста — один без отступа, второй — с левым отступом и шрифтом красного цвета. Для этого нужно создать правила для каждого из них, например так:

p {margin-left: 0;}

p.warn {margin-left: 40px; color: *#FF00;}*

Для применения созданного класса его имя нужно указать в атрибуте class для выбранных абзацев:

<p class=”warn”>Красный текст с отступом слева</p>

Общий синтаксис описания класса:

селектор.имя\_класса {описание}

При создании класса селектор можно не указывать, тогда это правило можно применять к любому селектору, поддерживающему тот же набор свойств.

Вот несколько примеров:

Правило:

.solid\_blue {color: blue;}

Использование:

<p class=”solid\_blue”>Синий текст абзаца</p>

<li class=”solid\_blue”>Синий текст элемента списка</li>

Правило:

h1.bigsans {font-family: Sans; font-size: 1.5em;}

h1.smallserif {font-family: Serif; font-size: .84em;}

Использование:

<h1 class=”bigsans”>Большой, но рубленый</h1>

<h1 class=”smallserif”>Маленький, но с засечками</h1>

**2.5 ВСТРОЕННЫЙ СТИЛИ**

Встраивание стилей предоставляет максимальный контроль над всеми элементами веб-страницы. Встроенный стиль применяется к любому тегу HTML с помощью атрибута style следующим образом:

<p style="font: 12pt Courier">Это текст с кеглем 12 точек и гарнитурой Courier</P>

Пример:

<div style="font-family: Garamond; font-size: 18 pt;>"

Весь текст в этом разделе имеет размер 18 точек и шрифт Garamond.

<span style="color:#ff3300;">

А этот фрагмент еще и выделен красным цветом.</span>

</div>

Встроенные стили полезны, когда необходима тонкая настройка отображения некоторого элемента страницы или небольшой веб-страницы.

**2.6 ВНЕДРЕННЫЕ СТИЛИ**

Внедренные стили используют тег <style>, который обычно размещают в заголовке HTML-документа (<head>...</head>):

<html>

<head>

...

<style>

правила CSS

</style>

...

</head>

<body>

...

**2.7 СВЯЗАННЫЕ ТАБЛИЦЫ СТИЛЕЙ**

Связанные (linked), или внешние (external) таблицы стилей — наиболее удобное решение, когда речь идет об оформлении целого сайта. Описание правил помещается в отдельный файл (обычно, но не обязательно, с расширением .css). С помощью тега <link> выполняется связывание этой таблицы стилей с каждой страницей, где ее необходимо применить, например так:

<link rel=stylesheet href="sample.css" type="text/css">

Любая страница, содержащая такую связь, будет оформлена в соответствии со стилями, указанными в файле sample.css. Следует отметить, что файл со стилями физически может находиться на другом веб-сервере, тогда в href нужно указать абсолютный путь к нему.

## **2.8 АППАРАТНО-ЗАВИСИМЫЕ СТИЛИ**

Таблицы стилей могут применяться для управления отображением содержимого в зависимости от используемого устройства вывода (монитор, проектор, устройство печати, звуковой синтезатор и т.п.). Для этого в описание стилей включить тип устройства, например так:

**@media** print {/\* печатающее устройство \*/

BODY { font-size: 10pt; }

}

**@media** screen { /\* монитор \*/

BODY { font-size: 12pt; }

}

**@media** screen, print {

BODY { line-height: 1.2; }

}

**@media** all {

BODY { margin: 1pt; }

}

Как видно из примера, вся таблица разбивается на секции, каждая из которых начинается со слова @media, за которым следует название класса устройств и далее, в фигурных скобках, непосредственно описание стилей.

Можно разделить таблицы стилей иначе, указав тип устройства в теге <link>:

<link rel=stylesheet href="sample.css" type="text/css" media=”screen”>

**2.9 ИДЕНТИФИКАТОРЫ**

В качестве селектора может выступать идентификатор элемента гипертекста, указанный в атрибуте id. Для назначения стилей таким элементам используется синтаксис, аналогичный описанию классов, но вместо точки ставится знак # (“решетка”). Например:

div*#content {*

position: absolute;

top: 10px;

left: 10%;

right: 10%;

border: solid 1px silver;

}

...

<div id="content">Текст</div>

Следует помнить, что идентификаторы элементов должны быть уникальны в пределах документа.

## **2.10 СВОЙСТВА CSS**

В таблице ниже перечислены некоторые часто используемые свойства CSS и их назначение.

|  |  |  |
| --- | --- | --- |
| **Имя** | **Значения** | **Описание** |
| background | [background-color || background-image || background-repeat || background-attachment || background-position] | inherit | Управление фоном элемента |
| background-color | <color> | transparent | inherit | Цвет фона |
| background-image | <uri> | none | inherit | Фоновое изображение |
| background-position | [ [<percentage> | <length> ]{1,2} | [ [top | center | bottom] || [left | center | right] ] ] | inherit | Положение фоновой картинки |
| background-repeat | repeat | repeat-x | repeat-y | no-repeat | inherit | Повторение фоновой картинки |
| border | [ border-width || border-style || <color> ] | inherit | Границы элемента |
| border-collapse | collapse | separate | inherit | Объединение/разделение смежных границ |
| border-color | <color>{1,4} | transparent | inherit | Цвет границы |
| border-style | <border-style>{1,4} | inherit | Стиль линии границы |
| border-top border-right border-bottom border-left | [ border-top-width || border-style || <color> ] | inherit | Управление стилем заданной границы |
| border-width | <border-width>{1,4} | inherit | Толщина линии границы |
| bottom | <length> | <percentage> | auto | inherit | Низ элемента |
| clear | none | left | right | both | inherit | Запрет заполнения свободного пространства рядом с элементом |
| clip | <shape> | auto | inherit | Обрезка содержимого элемента |
| color | <color> | inherit | Цвет содержимого |
| cursor | [ [<uri> ,]\* [ auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help ] ] | inherit | Форма курсора |
| display | inline | block | list-item | run-in | compact | marker | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit | Способ отображения элемента |
| empty-cells | show | hide | inherit | Отображение пустых ячеек таблицы |
| float | left | right | none | inherit | Свободное размещение элемента |
| font | [ [ font-style || font-variant || font-weight ]? font-size [ / line-height ]? font-family ] | caption | icon | menu | message-box | small-caption | status-bar | inherit | Управление шрифтом |
| font-family | [[ <family-name> | <generic-family> ],]\* [<family-name> | <generic-family>] | inherit | Гарнитура |
| font-size | <absolute-size> | <relative-size> | <length> | <percentage> | inherit | Кегль |
| font-style | normal | italic | oblique | inherit | Стиль шрифта |
| font-variant | normal | small-caps | inherit | Варианты отображения шрифта |
| font-weight | normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit | Толщина шрифта |
| height | <length> | <percentage> | auto | inherit | Ширина элемента |
| left | <length> | <percentage> | auto | inherit | Положение левой границы элемента |
| line-height | normal | <number> | <length> | <percentage> | inherit | Высота строки |
| list-style | [ list-style-type || list-style-position || list-style-image ] | inherit | Стиль списка |
| margin | <margin-width>{1,4} | inherit | Внешний отступ |
| margin-top margin-right margin-bottom margin-left | <margin-width> | inherit | Внешний отступ по заданной стороне |
| padding | <padding-width>{1,4} | inherit | Внутренний отступ |
| padding-top padding-right padding-bottom padding-left | <padding-width> | inherit | Внутренний отступ по заданной стороне |
| position | static | relative | absolute | fixed | inherit | Позиционирование элемента |
| right | <length> | <percentage> | auto | inherit | Положение правой границы |
| text-align | left | right | center | justify | <string> | inherit | Выравнивание текстового блока |
| text-decoration | none | [ underline || overline || line-through || blink ] | inherit | Текстовые эффекты |
| text-indent | <length> | <percentage> | inherit | Абзацный отступ |
| text-transform | capitalize | uppercase | lowercase | none | inherit | Начертание текста |
| top | <length> | <percentage> | auto | inherit | Положение верхней границы элемента |
| vertical-align | baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit | Вертикальное выравнивание в пределах блока |
| visibility | visible | hidden | collapse | inherit | Управление видимостью элемента |
| white-space | normal | pre | nowrap | inherit | Управление пробелами между словами |
| width | <length> | <percentage> | auto | inherit | Ширина элемента |
| z-index | auto | <integer> | inherit | Порядок перехода по клавише Tab |

## **ГЛАВА 3. СКРИПТЫ. JAVA SCRIPT**

## **3.1 ЧТО ТАКОЕ СКРИПТЫ И ИХ ВИДЫ**

**Скрипты** представляют собой специальные программы, предназначенные для обработки заданной информации и последующего взаимодействия с браузером с целью вывода полученного результата пользователю. **Скрипт** — это программа или программный файл сценарий, которые автоматизируют некоторую задачу, которую пользователь делал бы вручную, используя интерфейс программы. Скрипты пишутся на скриптовый языках, которые различаются по своему синтаксису, сферам применения и возможностям. Например: *AngelScript, Perl, Python, PHP, JavaScript, JScript* и другие.

**Сфера применения скриптов** огромна. Например:

* при их помощи пользователь получает возможность обращаться к базам данных
* seo-скрипты, помогающие продвигать сайты, эти скрипты написаны, используя специальные программы автоматизации браузера — например **XHE**.
* наблюдать статистику посещений (счетчики посещаемости)
* совершать записи в гостевых книгах
* оставлять комментарии к понравившимся статьям
* на скриптах основаны все **cms** и форумы
* скрипты помогают динамическому отображению веб-сайта
* скрипты позволяют организовать изменения части сайта без перегрузки всей страницы
* и многое многое другое

Основное отличие скрипта от программы состоит в том, что **script интерпретируется**, тогда как **программа выполняется** (т.е. исходный файл сначала компилируется, и результат этого ожидается компиляция).

[**Script**](http://en.wikipedia.org/wiki/Script_%28computing%29):

"Скрипт" отличается от основного код приложения, который обычно написан на другом языке и часто создается или минимально изменен конечным пользователем. Скрипты часто интерпретируются из исходного кода или байт-кода, тогда как приложения, которые они контролируют, традиционно составляются на машинном коде.

[**Программа**](http://en.wikipedia.org/wiki/Computer_program):

Программа имеет исполняемую форму, что компьютер может использовать как выполнение инструкции. Те же программы в своем человеко-читаемом источнике кода, из которого исполняемый файл программы производится (например, скомпилирован).

Все скрипты можно разделить на две большие группы: **исполняемые на стороне клиента** (в качестве которого выступает пользовательский компьютер) и **на стороне сервера**.

**Скрипты, исполняемые на стороне клиента (Client-side scripts)**  
Выполняются такие скрипты на компьютере пользователя посредством вызова определенных команд, которые обозначены в фрагментах кода обыкновенной веб-страницы. Для того чтобы они могли быть выполнены, браузер должен уметь их интерпретировать, то есть поддерживать язык, на котором скрипты написаны. В настоящее время существуют два языка для написания таких скриптов - **JavaScript и VBScript,** причем первый из них поддерживается практически всеми современными моделами браузеров, а второй – только Microsoft Internet Explorer. Скрипты на JavaScript и VBScript могут исполняться как автоматически во время загрузки страницы, так и при каких-либо действиях пользователя. Конкретный способ выполнения зависит от текста скрипта. Скрипт, выполняемый автоматически, состоит из простого набора команд, без каких-либо заголовков и окончаний.

Для того чтобы скрипт выполнялся при наступлении определенных условий или событий (нажатие пользователя на кнопку, окончание загрузки страницы и др.), необходимо оформить его как функцию. Это значит, что тогда у скрипта должен появиться заголовок и окончание.

Для JavaScript заголовком является оператор **function** имя\_скрипта()

В VBScript заголовок - слово **Sub** имя\_скрипта()

**Скрипты, выполняемые на стороне сервера (Server-side scripts)**К этой категории относятся **сценарии**, написанные на таких скриптовых языках, как Perl (CGI), ASP, PHP и JSP.

**CGI/Perl**

CGI (Common Gateway Interface, общий **шлюзовый интерфейс**) представляет собой часть веб-сервера, способную взаимодействовать с другими программами, и в этом смысле является **шлюзом для передачи** **данных**, полученных от клиента, программам-обработчикам информации – базам данных, электронным таблицам и пр. Чаще всего CGI-скрипты пишутся на языке Perl, хотя встречаются сценарии, написанные на C++, Pascal и др.

**ASP (Active Server Pages)**Язык ASP (Active Server Pages) был разработан компанией Microsoft и потому требует установки специального программного обеспечения Microsoft Internet Information Server. По большому счету ASP представляет собой совокупность средств программирования с использованием HTML, чтение и запись в базу данных посредством ODBC (Open DataBase Connection, открытое подключение к базе данных).  
Активные серверные страницы не требуют каких-либо специфических браузеров или иного ПО со стороны клиента. Все скрипты запускаются и выполняются на сервере, а браузер получает только результирующие HTML-файлы и отображает их для пользователя.

**PHP (Personal Home Page)**PHP - язык серверных скриптов, которые встраиваются в HTML-документы, и исполняясь на сервере, выдают результат обработки данных в окне браузера. Именно поэтому PHP называют **препроцессором** HTML (об этом даже гласит большая надпись на входе на официальный сайт этого языка. PHP является очень гибким языком, поддерживает MySQL, NNTP, POP3, IMAP, Oracle, PostgreSQL, dBase, ODBC, Informix и многие другие технологии. С его помощью можно реализовать все то же, что позволяет сделать CGI-программа, отличие заключается в том, что PHP более прост для понимания.

**JSP (Java Server Pages)**  
Java Server Pages являются ответом компании Sun Microsystems на язык ASP от Microsoft и представляют собой совокупность элементов и шаблонов данных, описывающих процесс обработки запросов. Эта технология позволяет использовать веб-программистам две основные возможности: обеспечение доступа к удаленным блокам данных и вычленение Java-кода из гипертекстовых фрагментов. В основе механизма действия JSP лежит обработка **XML-данных** и небольших сценариев, написанных на языке Java, для последующей генерации содержимого конечного документа.

**JavaScript** (принято произносить "Джаваскрипт") – это язык программирования, выполняющийся на стороне пользователя с помощью браузера. Он позволяет управлять элементами веб-страницы - заставлять их менять свои свойства и расположение, двигаться, реагировать на события, такие как перемещение мыши или нажатия клавиатуры, а также создавать множество других интересных эффектов.

## **3.2 ОСНОВНЫЕ ПОНЯТИЯ JAVA SCRIPT**

**JavaScript** часто сокращают до аббревиатуры **JS**, что часто используется в названиях различных фреймворков (Node.js, Ember.js) а также в расширении файлов с JavaScript-кодом.

**JavaScript** как название является зарегистрированной торговой маркой и принадлежит компании ORACLE.

**JavaScript** имеет синтаксис схожий с языком Си, однако имеет ряд существенных отличий:

 - Возможность работы с объектами, в том числе определение типа и структуры объекта во время выполнения программы.

 - Возможность передавать и возвращать функции как параметры, а также присваивать их переменной.

 - Наличие механизма автоматического приведения типов.

 - Автоматическая сборка мусора.

 - Использование анонимных функций.

**Преимущества:**

1. Ни один современный браузер не обходится без поддержки JavaScript.
2. С использованием написанных на JavaScript плагинов и скриптов справится даже не специалист.
3. Полезные функциональные настройки.
4. Постоянно совершенствующийся язык – сейчас разрабатывается бета-вариация проекта, JavaScript2.
5. Взаимодействие с приложением может осуществляется даже через текстовые редакторы – Microsoft Office и Open Office.
6. Перспектива использования языка в процессе обучения программированию и информатике.

**Недостатки:**

1. Пониженный уровень безопасности ввиду повсеместного и свободного доступа к исходным кодам популярных скриптов.
2. Множество мелких раздражающих ошибок на каждом этапе работы. Большая часть из них легко исправляется, но их наличие позволяет считать этот язык менее профессиональным, сравнительно с другими.
3. Повсеместное распространение. Своеобразным недостатком можно считать тот факт, что часть активно используемых программ (особенно приложений) перестанут существовать при отсутствии языка, поскольку целиком базируются на нем.

**Что может JavaScript в браузере?**

Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого.

Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, [Node.JS](https://ru.wikipedia.org/wiki/Node.js) поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д.

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

Например, в браузере JavaScript может:

* Добавлять новый HTML на страницу, изменять существующее содержимое, модифицировать стили.
* Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
* Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (**технологии**[**AJAX**](https://ru.wikipedia.org/wiki/AJAX)**и**[**COMET**](https://ru.wikipedia.org/wiki/Comet_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))).
* Получать и устанавливать **куки**, задавать вопросы посетителю, показывать сообщения.
* Запоминать данные на стороне клиента («local storage»).

**Чего НЕ может JavaScript в браузере?**

Возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесения ущерба данным пользователя.

Примеры таких ограничений включают в себя:

* JavaScript на веб-странице не может читать/записывать произвольные файлы на жёстком диске, копировать их или запускать программы. Он не имеет прямого доступа к системным функциям ОС.

Современные браузеры позволяют ему работать с файлами, но с ограниченным доступом, и предоставляют его, только если пользователь выполняет определённые действия, такие как «перетаскивание» файла в окно браузера или его выбор с помощью тега <input>.

Существуют способы взаимодействия с камерой/микрофоном и другими устройствами, но они требуют явного разрешения пользователя. Таким образом, страница с поддержкой JavaScript не может незаметно включить веб-камеру, наблюдать за происходящим и отправлять информацию в [ФСБ](https://ru.wikipedia.org/wiki/%D0%A4%D0%B5%D0%B4%D0%B5%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0_%D0%B1%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B9%D1%81%D0%BA%D0%BE%D0%B9_%D0%A4%D0%B5%D0%B4%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8).

* Различные окна/вкладки не знают друг о друге. Иногда одно окно, используя JavaScript, открывает другое окно. Но даже в этом случае JavaScript с одной страницы не имеет доступа к другой, если они пришли с разных сайтов (с другого домена, протокола или порта).

Это называется «Политика одинакового источника» (Same Origin Policy). Чтобы обойти это ограничение, обе страницы должны согласиться с этим и содержать JavaScript-код, который специальным образом обменивается данными.

Это ограничение необходимо, опять же, для безопасности пользователя. Страница http://anysite.com, которую открыл пользователь, не должна иметь доступ к другой вкладке браузера с URL http://gmail.com и воровать информацию оттуда.

* JavaScript может легко взаимодействовать с сервером, с которого пришла текущая страница. Но его способность получать данные с других сайтов/доменов ограничена. Хотя это возможно, требуется явное согласие (выраженное в заголовках HTTP) с удалённой стороной. Опять же, это ограничение безопасности.

**Итого:**

* JavaScript изначально создавался только для браузера, но сейчас используется на многих других платформах.
* Сегодня JavaScript занимает уникальную позицию в качестве самого распространённого языка для браузера, обладающего полной интеграцией с HTML/CSS.
* Многие языки могут быть **«транспилированы»** в JavaScript для предоставления дополнительных функций. Рекомендуется хотя бы кратко рассмотреть их после освоения JavaScript.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся [«движком» JavaScript](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA_JavaScript).

У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript».

Разные движки имеют разные «кодовые имена». Например:

* [V8](https://ru.wikipedia.org/wiki/V8_(%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA_JavaScript)) – в Chrome и Opera.
* [SpiderMonkey](https://ru.wikipedia.org/wiki/SpiderMonkey) – в Firefox.

**Как работают движки?**

Движки сложны. Но основы понять легко.

1. Движок (встроенный, если это браузер) читает текст скрипта.
2. Затем он преобразует («компилирует») скрипт в машинный язык.
3. После этого машинный код запускается и работает достаточно быстро.

Движок применяет оптимизации на каждом этапе. Он даже просматривает скомпилированный скрипт во время его работы, анализируя проходящие через него данные, и применяет оптимизации к машинному коду, полагаясь на полученные знания. В результате скрипты работают очень быстро.

|  |  |
| --- | --- |
| **Необходимые навыки:** | Базовая компьютерная грамотность, знание основ HTML и CSS. |
| **Цели:** | Знакомство с JavaScript и его возможностями, способами его подключения к веб-странице. |

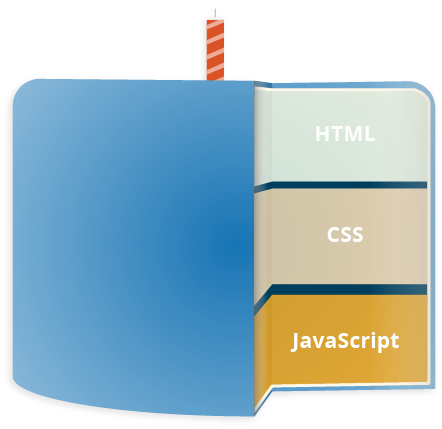
**Использование JavaScript в WEB**

Из-за своей простоты и удобства, JavaScript очень часто используется в мире WEB. По мере роста сложности сайтов и различных WEB-приложений, его использование начинается с динамического изменения элеметов сайта, до их генерации и разработки серверной части **(**[**клиент-серверная архитектура**](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80)**)** различных типов приложений. С помощью него создаются различные **фреймворки** и [библиотеки](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), что предлагают готовые решения типичных задач программирования в области WEB.

JavaScript чаще всего используется для интерактивного взаимодействия пользователя с интерфейсом приложений, когда при совершении тех или иных действий, приложение меняет свой стиль оформления, как бы взаимодействуя с пользователем. Такие реакции на действия пользователя, в пределах разумных норм, способны сделать взаимодействие с приложением более удобным и информативным.

JavaScript отличается так называем “низким порогом вхождения”, то есть для его изучения необходимо иметь не большой объем знаний в области WEB, в частности включающий базовые понятия [html](https://ru.wikipedia.org/wiki/HTML" \t "_blank)и [css](https://ru.wikipedia.org/wiki/CSS" \t "_blank).

* [HTML](https://developer.mozilla.org/ru/docs/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C/HTML) - это язык разметки, который мы используем для визуального и смыслового структурирования нашего web контента
* [CSS](https://developer.mozilla.org/ru/docs/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C/CSS) - это язык стилей с помощью которого мы придаем стиль отображения нашего HTML контента
* [JavaScript](https://developer.mozilla.org/ru/docs/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C/JavaScript) язык программирования, который позволяет Вам создать динамически обновляемый контент, управляет мультимедиа, анимирует изображения, впрочем, делает всё, что угодно



**Рис 2.2.1 Три слоя прекрасно выстраиваются друг над другом**

Для полноценного использования JavaScript вам необходимо пройти несколько шагов:

1. Изучение базового синтаксиса JavaScript - типы данных, переменные, операции, ветвление и циклы.

2. Освоение понятий функций, объектов и массивов, используемые в JavaScript.

3. Изучение **DOM** и управление им с помощью JavaScript. DOM (Document Object Model) - это объектная модель документа, благодаря которой можно работать с тегами HTML-документа как с деревом объектов, каждый из которых имеет свой уникальный адрес. Обращаясь по этому адресу с помощью JavaScript можно получить доступ к тому или иному HTML-объекту и управлять его свойствами - изменять размер, цвет и т.п.)

4. Изучение **BOM** (Browser Object Model) - структуры, имеющей схожее строение с DOM и включающее в себя элементы браузера - окна браузера, статусная строка, история просмотра и другие. Ими также можно управлять с помощью JavaScript.

5. После освоения уже чуть более сложных скриптов можно переходить к изучению и использованию основных JavaScript-библиотек, упрощающих взаимодействие JavaScript и HTML, таких как JQuery, Prototype.

**Сперва следует понять, для чего нужен JavaScript**

JavaScript (JS) – нативный язык, база, а библиотеки и JavaScript фреймворки – все то, что уже "накладывается" сверху. Язык программирования JavaScript клиентский и выполняется на стороне браузера. Грубо говоря, вся нагрузка ложится именно на ПК пользователя, а не на сервер, как было бы в случае с PHP. Поэтому кривой скрипт также будет затрагивать напрямую клиента, а не сервер: хорошо для вас, да плохо для пользователя.

**Определиться со стеком**

Не упускайте из виду JavaScript фреймворки, библиотеки и прочие инструменты.

* jQuery – дополнительная библиотека, способная облегчить работу с **XHR-запросами и селекторами.**
* Node.js представляет собой **серверную вариацию** JS.
* Gulp – автоматизация работ (напр., по сборке проекта).
* Webpack  заменяет собой и Gulp, и многие другие инструменты. Это сборщик модулей, который  позволяет собрать все js-файлы в необходимое кол-во пакетов, а также убедиться в правильности порядка собранных файлов.

**Что учить сначала?**

Теперь давайте условимся: идеальной последовательности не существует. Все напрямую зависит от того, чем вы занимаетесь и на что ориентированы: **фронтенд или full-stack.** Если первое, делайте меньший упор на взаимодействие с серверной частью, а если второе – напротив.

## **3.3 ВОЗМОЖНОСТИ JS**

Ядро языка JavaScript состоит из некоторого количества обычных возможностей, которые позволяют делать следующее:

* Хранить данные внутри переменных.
* Операции над фрагментами текстов (известными в программировании как "строки").
* Запускать код в соответствии с определенными событиями происходящими на web странице.

Так называемые **интерфейсы прикладного программирования (API)** предоставляют вам дополнительные сверхспособности для использования в вашем коде JavaScript.

**API - это готовые наборы блоков кода, которые позволяют разработчику реализовывать программы, которые в противном случае было бы трудно или невозможно реализовать**. Они делают то же самое для программирования, что готовые комплекты мебели делают для домашнего строительства - гораздо проще брать готовые панели и скручивать их вместе, чтобы сделать книжную полку, чем самому разрабатывать дизайн и тому подобное.

**ПРОБЛЕМЫ:**

**1. Проблема работы с типами данных в JavaScript**

Важнейшей концепцией, которую начинающие программисты осваивают очень рано, является идея переменных, контейнеров, которые хранят информацию во время работы программы. Проблема JavaScript заключается в том, что этот язык слишком вольно и неаккуратно обращается с переменными. Он позволяет делать то, что не выглядит правильным, и игнорирует очевидные нестыковки. Его халатность способна превращать незначительные опечатки в катастрофы, нарушающие работу программ.  
  
Вот пример кода, который обречён на провал:  
var myNumber = 100;

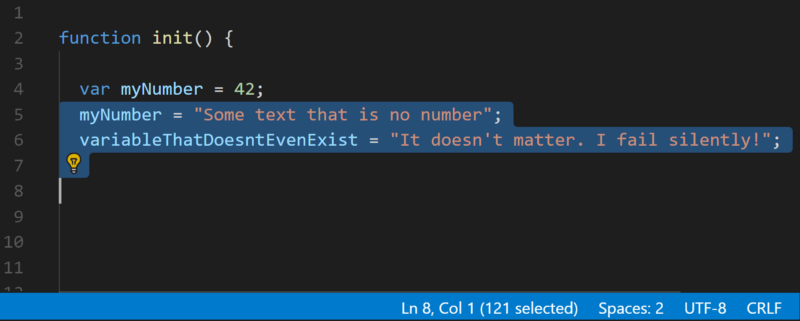
myNumber = myNumbr + 1;

Увидели ошибку? Во второй строке, на самом деле, используются две переменных: переменная myNumber, которая была объявлена в первой строке, и некая фиктивная переменная myNumbr. Мы знаем о том, что вторая переменная — это результат опечатки, но JavaScript сообщит нам об этом только во время выполнения программы, выдав сообщение об ошибке, которое выглядит как ReferenceError: myNumbr is not defined. То же самое происходит и в том случае, если при записи имени переменной перепутали большие и маленькие буквы (например, переменная из myNumber случайно превратилась в MyNumber). Проблема, но уже другого характера, может возникнуть и в том случае, если вы, считая, что некая переменная хранит число, прибавляете к ней другое число, а эта переменная, на самом деле, содержала строку или что-нибудь другое. JavaScript в таком случае не сообщит об ошибке, но результат вычислений будет совсем не таким, какого можно ожидать от сложения двух чисел

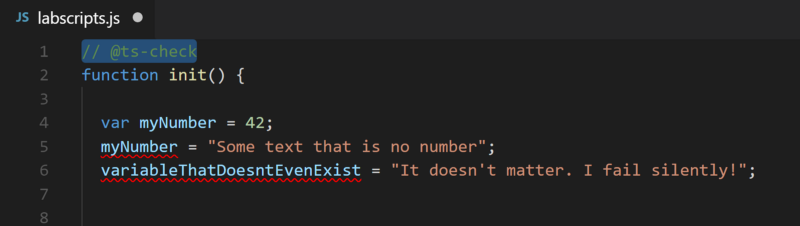
Достаточно сложно учиться программировать, и, в то же время, беспокоиться о том, как изучаемый язык программирования может вам навредить. Но, к счастью, проблемы безопасной работы с типами данных легко можно решить с помощью подходящих инструментов разработки.  
  
Один из редакторов кода, который подходит и для изучения программирования, и для профессионального использования — это [Visual Studio Code](https://code.visualstudio.com/). Он бесплатен, нетребователен к ресурсам, его код открыт, его можно практически до бесконечности расширять. Одна из его самых замечательных возможностей заключается в поддержке организации проверки JavaScript-кода на наличие в нём распространённых проблем средствами TypeScript. Включить такую проверку можно, воспользовавшись соответствующим конфигурационный файлом или вставив следующий комментарий в верхнюю часть JavaScript-файла:

*// @ts-check*

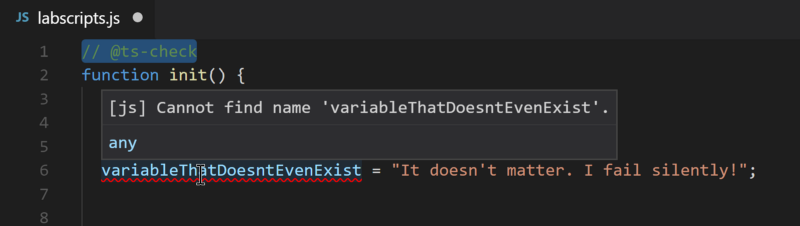
Вот пример, в котором имеются две ошибки, ни одна из которых, с точки зрения JavaScript, ошибкой не является.



**Проблемы, которые JavaScript не считает ошибками**  
  
Если добавить в начало этого файла комментарий // @ts-check, то VS Code выделит строки кода, которые содержат потенциальные ошибки.



**Выделенные строки кода**  
  
Если навести указатель мыши на выделенную строку, появится всплывающее окно, содержащее объяснение проблемы.



**Объяснение проблемы**  
  
Какие из этого можно сделать выводы? Хотя JavaScript толерантно относится ко многим нехорошим вещам, если скомбинировать JavaScript с качественным редактором кода и с правильными дополнительными инструментами, можно создать среду разработки, похожую на те, которыми обладают другие современные языки программирования. В результате разработка на JavaScript оказывается такой же удобной, как и на других языках, и, естественно, другие его преимущества никуда не деваются.

**2. Для JavaScript-разработки нужны дополнительные библиотеки и фреймворки**

Для того чтобы получить доступ к более широкому, чем есть в языке, набору функциональных возможностей (и не изобретать велосипеды), JavaScript-программистам нужно использовать библиотеки и фреймворки сторонних разработчиков.

## **3.4 РЕДАКТОРЫ КОДА**

Большую часть своего рабочего времени программисты проводят в редакторах кода. Есть два основных типа редакторов: IDE и «лёгкие» редакторы. Многие используют по одному инструменту каждого типа.

[**IDE**](http://learn.javascript.ru/code-editors#ide)

Термином [IDE](https://ru.wikipedia.org/wiki/Integrated_development_environment) (Integrated Development Environment, «интегрированная среда разработки») называют мощные редакторы с множеством функций, которые работают в рамках целого проекта. Как видно из названия, это не просто редактор, а нечто большее.

IDE загружает проект (который может состоять из множества файлов), позволяет переключаться между файлами, предлагает автодополнение по коду всего проекта (а не только открытого файла), также она интегрирована с **системой контроля версий** (например, такой как [git](https://git-scm.com/)), средой для тестирования и другими инструментами на уровне всего проекта.

Если вы ещё не выбрали себе IDE, присмотритесь к этим:

* [Visual Studio Code](https://code.visualstudio.com/) (бесплатно).
* [WebStorm](http://www.jetbrains.com/webstorm/) (платно).

Обе IDE – **кроссплатформенные.**

Для Windows есть ещё Visual Studio (не путать с Visual Studio Code). Visual Studio – это платная мощная среда разработки, которая работает только на Windows. Она хорошо подходит для .NET платформы. У неё есть бесплатная версия, которая называется [Visual Studio Community](https://www.visualstudio.com/vs/community/).

Многие IDE платные, но у них есть пробный период. Их цена обычно незначительна по сравнению с зарплатой квалифицированного разработчика, так что пробуйте и выбирайте ту, что вам подходит лучше других.

[**«Лёгкие» редакторы**](http://learn.javascript.ru/code-editors#lyogkie-redaktory)

«Лёгкие» редакторы менее мощные, чем IDE, но они отличаются скоростью, удобным интерфейсом и простотой.

В основном их используют для того, чтобы быстро открыть и отредактировать нужный файл.

Главное отличие между «лёгким» редактором и IDE состоит в том, что IDE работает на уровне целого проекта, поэтому она загружает больше данных при запуске, анализирует структуру проекта, если это необходимо, и так далее. Если вы работаете только с одним файлом, то гораздо быстрее открыть его в «лёгком» редакторе.

На практике «лёгкие» редакторы могут иметь множество плагинов, включая автодополнение и анализаторы синтаксиса **на уровне директории**, поэтому границы между IDE и «лёгкими» редакторами размыты.

Следующие варианты заслуживают вашего внимания:

* [Atom](https://atom.io/) (кроссплатформенный, бесплатный).
* [Sublime Text](http://www.sublimetext.com/) (кроссплатформенный, условно-бесплатный).
* [Notepad++](https://notepad-plus-plus.org/) (Windows, бесплатный).
* [Vim](http://www.vim.org/) и [Emacs](https://www.gnu.org/software/emacs/) тоже хороши, если знать, как ими пользоваться.

## **3.5 КНИГИ, СПРАВОЧНИКИ И КЛАССИФИКАЦИИ**

**Книги**

Книги по JavaScript для начинающих и аналогичные видеоуроки, которые разбудят в вас веб-разработчика. Массивы, [коллекции](https://proglib.io/p/javascript-sets/), строки и прочий базис обязательны к изучению на старте. Список литературы солидный, поэтому в первую очередь загляните сюда:

* [Изучаем программирование на JavaScript](https://www.ozon.ru/context/detail/id/31057355/)

<https://www.ozon.ru/context/detail/id/31057355/>

* [JavaScript и jQuery. Исчерпывающее руководство](https://t.me/progbook/620) (заодно и jQuery посмотрите)

<https://oz.by/books/more10336563.html>

* [Выразительный JavaScript](https://t.me/progbook/619)

<https://habr.com/ru/post/253101/>

[**Спецификация**](http://learn.javascript.ru/manuals-specifications#spetsifikatsiya)

[Спецификация ECMA-262](https://www.ecma-international.org/publications/standards/Ecma-262.htm) содержит самую глубокую, детальную и формализованную информацию о JavaScript. Она определяет сам язык.

Вначале спецификация может показаться тяжеловатой для понимания из-за слишком формального стиля изложения. Если вы ищете источник самой достоверной информации, то это правильное место, но она не для ежедневного использования.

Новая версия спецификации появляется каждый год. Между этими релизами, последний «черновик» спецификации доступен на <https://tc39.es/ecma262/>.

Чтобы почитать о самых последних возможностях, включая те, которые «почти в стандарте» (так называемые «stage 3 proposals»), посетите <https://github.com/tc39/proposals>.

Если вы разрабатываете под браузеры, то существуют и другие спецификации, о которых рассказывается во [второй части](http://learn.javascript.ru/browser-environment) этого учебника.

**[Справочники](http://learn.javascript.ru/manuals-specifications" \l "spravochniki)**

* **MDN (Mozilla) JavaScript Reference** – это справочник с примерами и другой информацией. Хороший источник для получения подробных сведений о функциях языка, методах встроенных объектов и так далее.

Располагается по адресу <https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference>.

Хотя зачастую вместо их сайта удобнее использовать какой-нибудь интернет-поисковик, вводя там запрос «MDN [что вы хотите найти]»,например <https://google.com/search?q=MDN+parseInt> для поиска информации о функции parseInt.

* **MSDN** – справочник от Microsoft, содержащий много информации, в том числе по JavaScript (который там часто обозначается как JScript). Если вам нужно найти что-то специфическое по браузеру Internet Explorer, лучше искать там: <http://msdn.microsoft.com/>.

Так же, как и в предыдущем случае, можно использовать интернет-поиск, набирая фразы типа «RegExp MSDN» или «RegExp MSDN jscript».

**[Таблицы совместимости](http://learn.javascript.ru/manuals-specifications" \l "tablitsy-sovmestimosti)**

JavaScript – это развивающийся язык, в который постоянно добавляется что-то новое.

Посмотреть, какие возможности поддерживаются в разных браузерах и других движках, можно в следующих источниках:

* [http://caniuse.com](http://caniuse.com/) – таблицы с информацией о поддержке по каждой возможности языка. Например, чтобы узнать, какие движки поддерживают современные **криптографические функции**, посетите: <http://caniuse.com/#feat=cryptography>.
* <https://kangax.github.io/compat-table> – таблица с возможностями языка и движками, которые их поддерживают и не поддерживают.

Все эти ресурсы полезны в ежедневной работе программиста, так как они содержат ценную информацию о возможностях использования языка, их поддержке и так далее.

Пожалуйста, запомните эти ссылки (или ссылку на эту страницу) на случай, когда вам понадобится подробная информация о какой-нибудь конкретной возможности JavaScript.

## **ГЛАВА 4. СТАНДАРТЫ**

## **4.1 КОМПАНИЯ, УТВЕРЖДАЮЩАЯ СТАНДАРТЫ HTML**

**WHATWG** (Рабочая группа по вебу, гипертексту, приложениям и технологиям, англ. *Web Hypertext Application Technology Working Group*) — сообщество специалистов по HTML и API, необходимым для веб-приложений, занимающееся развитием этих технологий и разработкой стандартов для них.

WHATWG была основана в 2004 году сотрудниками компаний — производителей браузеров: Apple, Mozilla Foundation и Opera Software (в лице Яна Хиксона; ныне работает в Google). В настоящий момент рабочая группа по-прежнему опирается на сообщество, а также на документы и политики, разработанные для WHATWG нынешними доминирующими производителями браузеров — Apple, [Google](https://ru.wikipedia.org/wiki/Google_(%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F)" \o "Google (компания)), Microsoft и Mozilla; четверо представителей этих четырёх организаций составляют руководящую группу WHATWG.

WHATWG разрабатывает стандарты для HTML, DOM, интерфейсов для получения ресурсов (стандарт Fetch), полноэкранного режима работы браузера и т. д. Спецификации WHATWG имеют статус «живых стандартов» (англ. *Living Standard*) — в спецификации постоянно вносятся правки и добавления. Документы WHATWG публикуются на условиях лицензии CC BY 4.0.

## **4.2 РОЛЬ В РАЗРАБОТКЕ СТАНДАРТОВ И ВЗАИМОДЕЙСТВИЕ С W3C**

До WHATWG единственной авторитетной организацией, вырабатывавшей стандарты для веба, был [W3C](https://ru.wikipedia.org/wiki/W3C) (Консорциум Всемирной паутины); в нём состоят сотни компаний, включая и связанные с WHATWG. Основатели WHATWG сочли, что W3C не учитывает в своей работе реальные потребности веб-разработчиков, в частности, что W3C слишком сосредоточился на XHTML вместо дальнейшего развития HTML. Поскольку WHATWG руководят представители производителей браузеров, которые в реальности занимаются реализацией стандартов в своих продуктах, влияние новой рабочей группы, оказалось, как минимум сопоставимо с влиянием W3C.

В конце 2000-х годов W3C и WHATWG совместно разработали HTML5, но дальнейшее развитие стандарта HTML организации стали осуществлять уже независимо. Сотрудничество между организациями постепенно становилось всё менее эффективным.

Чтобы избежать появления двух конкурирующих, взаимно противоречивых стандартов, W3C и WHATWG договорились о новом формате совместной работы. А именно, согласно подписанному обеими организациями в 2019 году **меморандуму**, ведущая роль в работе над стандартами HTML и DOM принадлежит WHATWG, а W3C приглашает своих членов участвовать в работе WHATWG и утверждает стандарты WHATWG как свои официальные рекомендации.

## **4.3 ВЕРСИИ**

* HTML 2.0 — опубликован [IETF](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%BD%D1%8B%D0%B9_%D1%81%D0%BE%D0%B2%D0%B5%D1%82_%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82%D0%B0)(Инженерный Совет Интернета) как [RFC 1866](https://tools.ietf.org/html/rfc1866) в статусе *Proposed Standard* ([24 ноября](https://ru.wikipedia.org/wiki/24_%D0%BD%D0%BE%D1%8F%D0%B1%D1%80%D1%8F) [1995 года](https://ru.wikipedia.org/wiki/1995_%D0%B3%D0%BE%D0%B4));
* HTML 3.0 — 28 марта 1995 года — IETF Internet Draft (до 28 сентября 1995 года);
* HTML 3.2 — [14 января](https://ru.wikipedia.org/wiki/14_%D1%8F%D0%BD%D0%B2%D0%B0%D1%80%D1%8F) [1997 года](https://ru.wikipedia.org/wiki/1997_%D0%B3%D0%BE%D0%B4);
* HTML 4.0 — [18 декабря](https://ru.wikipedia.org/wiki/18_%D0%B4%D0%B5%D0%BA%D0%B0%D0%B1%D1%80%D1%8F) 1997 года;
* HTML 4.01 — [24 декабря](https://ru.wikipedia.org/wiki/24_%D0%B4%D0%B5%D0%BA%D0%B0%D0%B1%D1%80%D1%8F) [1999 года](https://ru.wikipedia.org/wiki/1999_%D0%B3%D0%BE%D0%B4);
* ISO/IEC 15445:2000 (так называемый ISO HTML, основан на HTML 4.01 Strict) — [15 мая](https://ru.wikipedia.org/wiki/15_%D0%BC%D0%B0%D1%8F) [2000 года](https://ru.wikipedia.org/wiki/2000_%D0%B3%D0%BE%D0%B4);
* [HTML5](https://ru.wikipedia.org/wiki/HTML5) — [28 октября](https://ru.wikipedia.org/wiki/28_%D0%BE%D0%BA%D1%82%D1%8F%D0%B1%D1%80%D1%8F) [2014 года](https://ru.wikipedia.org/wiki/2014_%D0%B3%D0%BE%D0%B4);
* HTML 5.1 начал разрабатываться 17 декабря 2012 года. Рекомендован к применению с 1 ноября 2016 года.
* HTML 5.2 был представлен 14 декабря 2017 года.
* HTML 5.3 был представлен 24 декабря 2018 года.

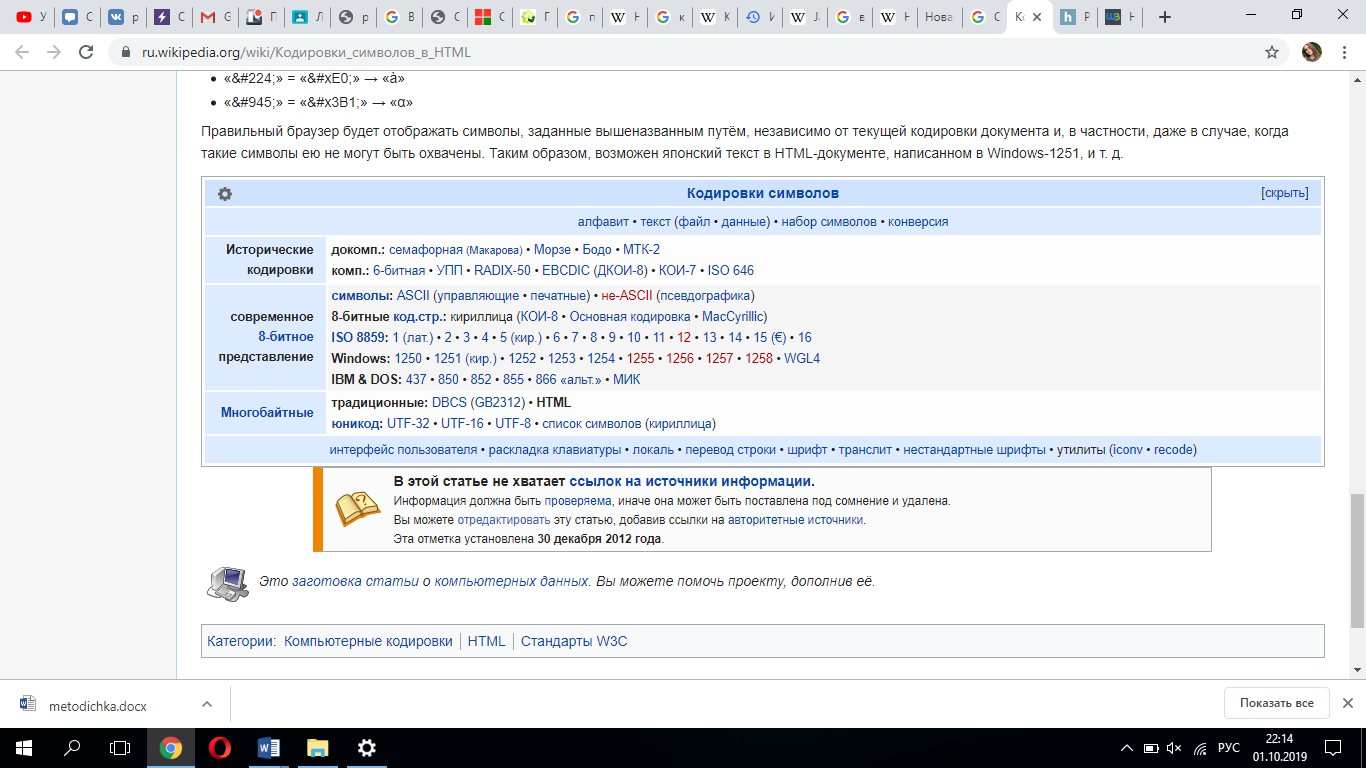
Официальной спецификации HTML 1.0 не существует. До 1995 года существовало множество неофициальных стандартов HTML. Чтобы стандартная версия отличалась от них, ей сразу присвоили второй номер.

Версия 3 была предложена [Консорциумом Всемирной паутины](https://ru.wikipedia.org/wiki/W3C) (W3C) в марте 1995 года и обеспечивала много новых возможностей, таких как создание таблиц, «обтекание» изображений текстом и отображение сложных [математических формул](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0), поддержка [gif формата](https://ru.wikipedia.org/wiki/GIF" \o "GIF). Даже при том, что этот стандарт был совместим со второй версией, реализация его была сложна для браузеров того времени. Версия 3.1 официально никогда не предлагалась, и следующей версией стандарта HTML стала 3.2, в которой были опущены многие нововведения версии 3.0, но добавлены нестандартные элементы, поддерживаемые браузерами [Netscape Navigator](https://ru.wikipedia.org/wiki/Netscape_Navigator" \o "Netscape Navigator) и [Mosaic](https://ru.wikipedia.org/wiki/Mosaic" \o "Mosaic).

В версии HTML 4.0 произошла некоторая «очистка» стандарта. Многие элементы были отмечены как устаревшие и не рекомендованные ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *deprecated*). В частности, тег <font>, используемый для изменения свойств [шрифта](https://ru.wikipedia.org/wiki/%D0%A8%D1%80%D0%B8%D1%84%D1%82), был помечен как устаревший (вместо него рекомендуется использовать таблицы стилей [CSS](https://ru.wikipedia.org/wiki/CSS)). В [1998 году](https://ru.wikipedia.org/wiki/1998_%D0%B3%D0%BE%D0%B4) [Консорциум Всемирной паутины](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D0%BE%D1%80%D1%86%D0%B8%D1%83%D0%BC_%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%BE%D0%B9_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D1%8B) начал работу над новым языком разметки, основанным на HTML 4, но соответствующим синтаксису XML. Впоследствии новый язык получил название [XHTML](https://ru.wikipedia.org/wiki/XHTML). Первая версия XHTML 1.0 одобрена в качестве Рекомендации консорциума Всемирной паутины [26 января](https://ru.wikipedia.org/wiki/26_%D1%8F%D0%BD%D0%B2%D0%B0%D1%80%D1%8F) [2000 года](https://ru.wikipedia.org/wiki/2000_%D0%B3%D0%BE%D0%B4).

Планируемая версия [XHTML 2.0](https://ru.wikipedia.org/wiki/XHTML_2.0) должна была разорвать совместимость со старыми версиями HTML и XHTML, но 2 июля [2009 года](https://ru.wikipedia.org/wiki/2009_%D0%B3%D0%BE%D0%B4) [Консорциум Всемирной паутины](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D0%BE%D1%80%D1%86%D0%B8%D1%83%D0%BC_%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%BE%D0%B9_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D1%8B) объявил, что полномочия рабочей группы XHTML2 истекают в конце [2009 года](https://ru.wikipedia.org/wiki/2009_%D0%B3%D0%BE%D0%B4). Таким образом, была приостановлена вся дальнейшая разработка стандарта XHTML 2.0.

## **4.4 КОДИРОВКА**



**Кодировка Unicode**

В 1991 году был разработан универсальный стандарт кодирования Unicode Transformatiддon Format (UTF), благодаря которому стало возможным представить знаки практически всех известных письменных языков. Разработчиками стандарта была некоммерческая организация Unicode Consortium.

Первой была разработана кодировка UTF-32, затем UTF-16, и наконец появился стандарт UTF-8.

В UTF-32 для кодирования одного символа использовалось 32 бита (4 байта), что увеличивало вес конечного файла в 4 раза.

В UTF-16 вес снизился вдвое, так как для кодирования использовалось уже 16 бит, но и это было далеко не оптимальным решением. Разработанный впоследствии стандарт UTF-8 на сегодняшний день считается самым оптимальным. Для кодирования одного символа в UTF-8 используется от 1 до 4 байт. Он идеально совмещается с системами, которые используют 8-битные символы.

UTF-8 унаследовал базовую часть кодировки ASCII (128 символов), то есть для кодировки латинских символов используется 8 бит (1 байт). Все последующие символы кодируются двумя байтами и более.

HTML4 поддерживает только кодировку UTF-8. В HTML5 поддерживаются как UTF-8, так и UTF-16.

## **ВОПРОСЫ ПО ИЗУЧЕННОМУ МАТЕРИАЛУ:**

1. Предоставьте полное название компании, занимающийся стандартами для html?

2. Расшифруйте название компании WHATWG?

3. Что за организация вырабатывала стандарты для веба до WHATWG?

4. С какой проблемой столкнулись веб-разработчики компании WHATWG и W3C?

5. Какая первая версия была предложена Консорциумом Всемирной паутины, которая поддерживала gif формат?

6. В версии какого стандарта произошла некоторая «очистка»?

7. Как называется язык разметки, основанный на HTML 4, но соответствующий синтаксису XML?

8. Как вы понимаете термин «Кодировка»?

9. Что вы знаете о стандарте кодирования Unicode Transformatiддon Format (UTF)?

10. Какой стандарт лучше: UTF-32 или UTF-16? Объясните почему?

11. Какой стандарт лучше: UTF-16 или UTF-8? Объясните почему?

12. Закончите предложение:

Для кодирования одного символа в UTF-8 используется от …. до …. байт.

13. Закончите предложение:

В UTF-32 для кодирования одного символа использовалось 32 бита (….. байта), что увеличивало вес конечного файла в …. раза.

14. Какая версия HTML поддерживает ТОЛЬКО кодировку UTF-8?

15. В чём отличие скриптов от программы?  
16. Что такое скрипт?  
17. На какие группы делятся скрипты?  
18. Как нужно оформить скрипт, чтобы он выполнился при наступлении определенных условий и событий? Как нужно оформить это в JS?  
19. Что такое PHP?  
20. Какой компании принадлежит JS?  
21. Что такое JavaScript2?  
22. Что может JS в браузере?  
23. Какое кодовое имя движка в Google Chrome и Opera?  
24. Что такое DOM и BOM?  
25. Какая библиотека JS может облегчить работу XHR-запросами и селекторами?  
26. Какие проблемы могут возникнуть при работе с JS? (опишите подробно )  
27. Назовите два основных редактора для работы с JS?  
28. Приведите примеры кроссплатформенных редакторов.  
29. Какая спецификация содержит самую глубокую детальную формализованную информацию о JS?  
30. Назовите два основных справочника, которые используются при работе с JS?

31.Что такое CSS?

32.Закончите определение: Целью создания CSS было …

33.Как выглядит CSS файл?

34.Какое расширение css файлов?

35.Что такое каскадная таблица стилей?

36.Какой тег используют внедренные стили?

37. За что отвечает свойство font?

38.Чем отличаются действия свойств display:none и visiblity:hidden?

39. Как расшифровывается HTML?  
40. Для чего используется HTML?  
41. Что такое тэги и для чего они нужны?  
42. С помощью какого тэга определяются заголовки?  
43. Для чего нужны комментарии ?  
44. С помощью каких редакторов можно сделать веб-страницу?  
45. Какие расширения используется при сохранении html-страницы ?

## **ИСТОЧНИКИ:**

1.Что  такое  JavaScript?  [Электронный ресурс]-  Режим  доступа: <https://www.ipipe.ru/info/javascript.html>

2.Современный учебник JavaScript [Электронный ресурс]-Режим доступа: <http://learn.javascript.ru/code-editors>

3.В чем разница между языками программирования JAVA и JavaScript? [Электронный ресурс]-Режим доступа: <https://star-marketing.com.ua/blog/v-chem-raznitsa-mezhdu-yazykami-programmirovaniya-java-i-javascript/>

4. JavaScript — лучший язык программирования для начинающих.Так это или нет?[Электронный ресурс]-Режим доступа:<https://medium.com/webbdev/js-240edd8f3887>

5. Что такое JavaScript? [Электронный ресурс]-Режим доступа: <https://developer.mozilla.org/ru/docs/Learn/JavaScript/%D0%9F%D0%B5%D1%80%D0%B2%D1%8B%D0%B5_%D1%88%D0%B0%D0%B3%D0%B8/What_is_JavaScript>

6. Лайфхак: в какой последовательности изучать JavaScript [Электронный ресурс]- Режим доступа:<https://proglib.io/p/sequence-of-study-javascript/>

7. 16 бесплатных книг по JavaScript [Электронный ресурс]-Режим доступа: <https://proglib.io/p/free-js-books/>

8.Официальный сайт WHATWG[Электронный ресурс]-Режим доступа:<https://html.spec.whatwg.org/multipage/>

9. HTML 4.0[Электронный ресурс]-Режим доступа: <https://www.w3.org/TR/html401/>

10.Статья из Википедии(основная информация об HTML)[Электронный ресурс]-Режим доступа: <https://ru.wikipedia.org/wiki/HTML>

11.Каскадные таблицы стилей [Электронный ресурс]- Режим доступа: <https://iit-web-lectures.readthedocs.io/ru/latest/www/css.html>

12.Html, CSS, JavaScript, SQL – что и зачем [Электронный ресурс]- Режим доступа: <http://codeharmony.ru/materials/125>

13. HTML. The language for building web pages [Электронный ресурс]- Режим доступа: [https://www.w3schools.com](https://www.w3schools.com/)