Основы языка Javascript

Про язык JavaScript и окружение для разработки на нём.

Оглавление

[Синтаксис языка 0](#_Toc444164282)

[Внешние скрипты и порядок их исполнения 1](#_Toc444164283)

[Структура кода 2](#_Toc444164284)

[Комментарии 2](#_Toc444164285)

[Переменные 3](#_Toc444164286)

[Константы 3](#_Toc444164287)

[Типы данных 4](#_Toc444164288)

[Оператор typeof 5](#_Toc444164289)

[Современный стандарт, "use strict" 6](#_Toc444164290)

[Домашнее задание 6](#_Toc444164291)

[Используемая литература 7](#_Toc444164292)

# Синтаксис языка

Программы на языке JavaScript можно вставить в любое место HTML при помощи тега SCRIPT. Например:

|  |
| --- |
| <head>  <title>Hello, JavaScript!</title>  <meta content="text/html; charset=Windows-1251" http-equiv="content-type"/>  <script language="javascript">  alert('Hello, JavaScript!');  </script>  </head> |

Этот пример использует следующие элементы:

**<script> ... </script>**

Тег script содержит исполняемый код. Предыдущие стандарты HTML требовали обязательного указания атрибута type, но сейчас он уже не нужен. Достаточно просто <script>.

Браузер, когда видит **<script>:**

1. Начинает отображать страницу, показывает часть документа до script
2. Встретив тег script, переключается в JavaScript-режим и не показывает, а исполняет его содержимое.
3. Закончив выполнение, возвращается обратно в HTML-режим и только тогда отображает оставшуюся часть документа.

В старых скриптах оформление тега SCRIPT было немного сложнее. В устаревших руководствах можно встретить следующие элементы:

Атрибут **<script type=…>**

В отличие от HTML5, стандарт HTML 4 требовал обязательного указания этого атрибута. Выглядел он так: type="text/javascript". Если указать другое значение type, то скрипт выполнен не будет.

В современной разработке атрибут type не обязателен.

Атрибут **<script language=…>**

Этот атрибут предназначен для указания языка, на котором написан скрипт. По умолчанию, язык – JavaScript, так что и этот атрибут ставить не обязательно.

# Внешние скрипты и порядок их исполнения

Если JavaScript-кода много – его выносят в отдельный файл, который подключается в HTML:

**<script src="/path/to/script.js"></script>**

Здесь **/path/to/script.js** – это абсолютный путь к файлу, содержащему скрипт (из корня сайта).

Браузер сам скачает скрипт и выполнит.

Можно указать и полный URL, например:

**<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.3.0/lodash.js"></script>**

Вы также можете использовать путь относительно текущей страницы. Например, **src="lodash.js"** обозначает файл из текущей директории.

Чтобы подключить несколько скриптов, используйте несколько тегов:

**<script src="/js/script1.js"></script>**

**<script src="/js/script2.js"></script>**

...

# Структура кода

Все команды на языка Javascript разделяются точкой с запятой.

|  |
| --- |
| alert('Привет'); alert('Мир'); |

Со временем программа становится большой и сложной. Появляется необходимость добавить комментарии, которые объясняют, что происходит и почему.

## Комментарии

Комментарии могут находиться в любом месте программы и никак не влияют на её выполнение. Интерпретатор JavaScript попросту игнорирует их.

Однострочные комментарии начинаются с двойного слэша //. Текст считается комментарием до конца строки:

|  |
| --- |
| // Команда ниже говорит "Привет"  alert( 'Привет' );  alert( 'Мир' ); // Второе сообщение выводим отдельно |

Многострочные комментарии начинаются слешем-звездочкой «/\*» и заканчиваются звездочкой-слэшем «\*/», вот так:

|  |
| --- |
| /\* Пример с двумя сообщениями.  Это - многострочный комментарий.  \*/  alert( 'Привет' );  alert( 'Мир' ); |

Всё содержимое комментария игнорируется. Если поместить код внутрь /\* … \*/ или после // – он не выполнится.

|  |
| --- |
| /\* Закомментировали код  alert( 'Привет' );  \*/  alert( 'Мир' ); |

В большинстве редакторов комментарий можно поставить горячей клавишей, обычно это **Ctrl+/** для однострочных и что-то вроде **Ctrl+Shift+/** – для многострочных комментариев (нужно выделить блок и нажать сочетание клавиш). Детали смотрите в руководстве по редактору.

## Переменные

Переменная состоит из имени и выделенной области памяти, которая ему соответствует.

Для объявления или, другими словами, создания переменной используется ключевое слово var:

|  |
| --- |
| var message; |

После объявления, можно записать в переменную данные:

|  |
| --- |
| var message;  message = 'Hello'; // сохраним в переменной строку |

Эти данные будут сохранены в соответствующей области памяти и в дальнейшем доступны при обращении по имени:

|  |
| --- |
| var message;  message = 'Hello!';  alert( message ); // выведет содержимое переменной |

Для краткости можно совместить объявление переменной и запись данных:

|  |
| --- |
| var message = 'Hello!'; |

Можно даже объявить несколько переменных сразу:

|  |
| --- |
| var user = 'John', age = 25, message = 'Hello'; |

На имя переменной в JavaScript наложены всего два ограничения.

1. Имя может состоять из: букв, цифр, символов $ и \_
2. Первый символ не должен быть цифрой.

## Константы

Константа – это переменная, которая никогда не меняется. Как правило, их называют большими буквами, через подчёркивание. Например:

|  |
| --- |
| var COLOR\_RED = "#F00";  var COLOR\_GREEN = "#0F0";  var COLOR\_BLUE = "#00F";  var COLOR\_ORANGE = "#FF7F00";  var color = COLOR\_ORANGE;  alert( color ); // #FF7F00 |

Технически, константа является обычной переменной, то есть её можно изменить. Но мы договариваемся этого не делать.

Зачем нужны константы? Почему бы просто не писать var color = "#FF7F00"?

1. Во-первых, константа COLOR\_ORANGE – это понятное имя. По присвоению var color="#FF7F00" непонятно, что цвет – оранжевый. Иными словами, константа COLOR\_ORANGE является «понятным псевдонимом» для значения #FF7F00.
2. Во-вторых, опечатка в строке, особенно такой сложной как #FF7F00, может быть не замечена, а в имени константы её допустить куда сложнее.

## Типы данных

1. **Число «number»**

|  |
| --- |
| var n = 123;  n = 12.345; |

Единый тип число используется как для целых, так и для дробных чисел.

Существуют специальные числовые значения Infinity (бесконечность) и NaN (ошибка вычислений).

Например, бесконечность Infinity получается при делении на ноль:

|  |
| --- |
| alert( 1 / 0 ); // Infinity |

Ошибка вычислений NaN будет результатом некорректной математической операции, например:

|  |
| --- |
| alert( "нечисло" \* 2 ); // NaN, ошибка |

Эти значения формально принадлежат типу «число», хотя, конечно, числами в их обычном понимании не являются.

1. **Строка «string»**

|  |
| --- |
| var str = "Мама мыла раму";  str = 'Одинарные кавычки тоже подойдут'; |

В JavaScript одинарные и двойные кавычки равноправны. Можно использовать или те или другие.

1. **Булевый (логический) тип «boolean»**

У него всего два значения: true (истина) и false (ложь).

Как правило, такой тип используется для хранения значения типа да/нет, например:

|  |
| --- |
| var checked = true; // поле формы помечено галочкой  checked = false; // поле формы не содержит галочки |

О нём мы поговорим более подробно, когда будем обсуждать логические вычисления и условные операторы.

1. **Специальное значение «null»**

Значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения null:

|  |
| --- |
| var age = null; |

В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».

В частности, код выше говорит о том, что возраст age неизвестен.

1. **Специальное значение «undefined»**

Значение undefined, как и null, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено».

Если переменная объявлена, но в неё ничего не записано, то её значение как раз и есть undefined:

|  |
| --- |
| var x;  alert( x ); // выведет "undefined" |

Можно присвоить undefined и в явном виде, хотя это делается редко:

|  |
| --- |
| var x = 123;  x = undefined;  alert( x ); // "undefined" |

В явном виде undefined обычно не присваивают, так как это противоречит его смыслу. Для записи в переменную «пустого» или «неизвестного» значения используется null.

1. Объекты «object»

Первые 5 типов называют «примитивными».

Особняком стоит шестой тип: «объекты».

Он используется для коллекций данных и для объявления более сложных сущностей.

Объявляются объекты при помощи фигурных скобок {...}, например:

|  |
| --- |
| var user = { name: "Вася" }; |

## Оператор typeof

Оператор typeof возвращает тип аргумента.

У него есть два синтаксиса: со скобками и без:

* Синтаксис оператора: typeof x.
* Синтаксис функции: typeof(x).

Работают они одинаково, но первый синтаксис короче. Результатом typeof является строка, содержащая тип:

|  |
| --- |
| typeof undefined // "undefined"  typeof 0 // "number"  typeof true // "boolean"  typeof "foo" // "string"  typeof {} // "object"  typeof null // "object" (1)  typeof function(){} // "function" (2) |

Последние две строки помечены, потому что typeof ведет себя в них по-особому.

1. Результат typeof null == "object" – это официально признанная ошибка в языке, которая сохраняется для совместимости. На самом деле null – это не объект, а отдельный тип данных.
2. Функции мы пройдём чуть позже. Пока лишь заметим, что функции не являются отдельным базовым типом в JavaScript, а подвидом объектов. Но typeof выделяет функции отдельно, возвращая для них "function". На практике это весьма удобно, так как позволяет легко определить функцию.

# Современный стандарт, "use strict"

Очень долго язык JavaScript развивался без потери совместимости. Новые возможности добавлялись в язык, но старые – никогда не менялись, чтобы не «сломать» уже существующие HTML/JS-страницы с их использованием.

Однако, это привело к тому, что любая ошибка в дизайне языка становилась «вмороженной» в него навсегда.

Так было до появления стандарта ECMAScript 5 (ES5), который одновременно добавил новые возможности и внёс в язык ряд исправлений, которые могут привести к тому, что старый код, который был написан до его появления, перестанет работать.

Чтобы этого не случилось, решили, что по умолчанию эти опасные изменения будут выключены, и код будет работать по-старому. А для того, чтобы перевести код в режим полного соответствия современному стандарту, нужно указать специальную директиву use strict.

Эта директива не поддерживается IE9-.

Директива выглядит как строка "use strict"; или 'use strict'; и ставится в начале скрипта:

|  |
| --- |
| "use strict";  // этот код будет работать по современному стандарту ES5  ... |

# Домашнее задание

* Вывести «Привет, Javascript» c помощью alert
* Работа с переменными
* Объявите две переменные: admin и name.
* Запишите в name строку "Василий".
* Скопируйте значение из name в admin.
* Выведите admin (должно вывести «Василий»).
* \* Самостоятельно разобраться с атрибутами тэга script (async и defer)

# Используемая литература

* [Mozilla Developer Network](https://developer.mozilla.org/)
* [MSDN](http://msdn.microsoft.com/)
* [Safari Developer Library](https://developer.apple.com/library/safari/navigation/index.html)
* [Современный учебник JavaScript](http://learn.javascript.ru)