
Урок 1

# Основы языка JavaScript

Общее знакомство с JavaScript, создание первого кода и его запуск.

# Введение

JavaScript — основной язык программирования в интернете. Он позволяет расширить стандартное поведение пользователя на веб-страницах. С JavaScript  страницы из сухих и статичных, которые просто показываются друг за другом, становятся интерактивными и взаимодействуют с пользователями, откликаются на их действия. Можно получать и применять данные из Сети и делать многое другое.

JavaScript схож с Java только названием — их не стоит путать. JavaScript был создан на пике популярности Java, что позволило разработчикам успешно применить элементы синтаксиса языков семейства C. В целом Java и JavaScript совершенно разные.

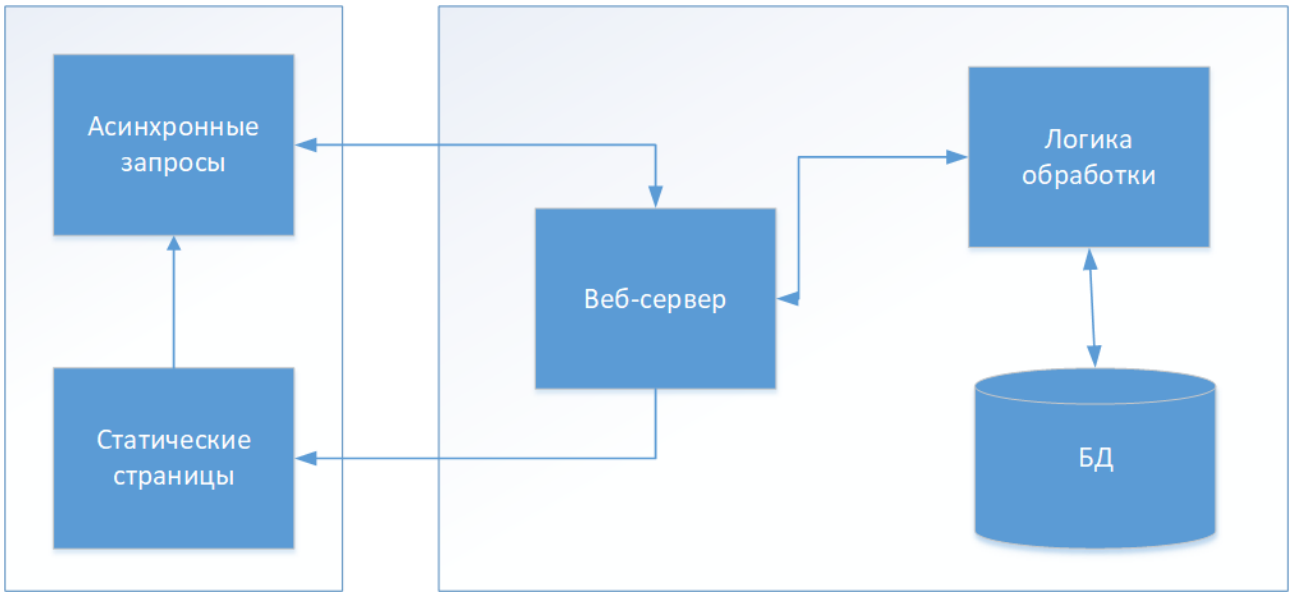
С опытом вы вполне сможете программировать на страницах не шаблонное, а абсолютно новое отображение реакций на действия пользователя. Не так давно для решения таких задач предпочтительной была технология Flash. Но с выходом HTML5 она потеряла актуальность из-за громоздкости и уязвимости. В HTML5 JavaScript стал стандартным языком сценариев.

Поскольку JavaScript входит в топ-10 языков программирования, ваши знания и опыт будут востребованы. JavaScript поддерживается всеми современными браузерами, поэтому на нем легко запускать программы. Сейчас это не только язык программирования веб-интерфейсов: все больше серверных реализаций JavaScript набирают популярность.

# Принципы работы JavaScript

Вы уже умеете создавать HTML-страницы с их структурой, наполнять их, верстать с макетов и применять стили. Но пока они не обладают уникальным поведением: умеют только предоставлять навигацию по ссылкам, реагировать на наведение мышки и подобное.

Но к поведению веб-страниц требований больше: они должны быть интерактивными, взаимодействовать с пользователем. В игру вступает JavaScript: посмотрим, какое место он занимает при оформлении поведения веб-страниц.




1. HTML определяет контент страниц, их структуры.

2. CSS определяет стиль оформления страницы.
3. Сервер обрабатывает HTTP-запросы, при необходимости передавая их интерпретатору.
4. Интерпретатор формирует HTML-документ, при необходимости обращаясь к БД.

JavaScript на этой схеме новичок. Он умеет определять поведение страниц. Например, странице нужно реагировать, когда пользователь кликает по кнопке «Отправить заказ». Появляется возможность наращивать программный код, выполняющий динамические операции.

# Принципы написания кода на JavaScript

Если вы уже знакомы с языками программирования, то знаете, что программа появляется в результате вполне определенных шагов. Сначала пишется код, при необходимости он компилируется, собирается в пакет, устанавливается на компьютер. JavaScript по циклу разработки отличается от данной схемы. Разработчик подключает код JavaScript к странице, которая загружается в браузер. После этого уже сам браузер делает все необходимое, чтобы выполнить код.

1. Страница создается по стандартной схеме. Верстается структура HTML, к ней применяются стили и добавляется содержимое.
2. К странице подключается код JavaScript. На примере вы увидите, что это происходит так же, как подключение файла CSS — код можно вынести в отдельный файл. Но как и с CSS, никто не запрещает писать JavaScript-код внутри разметки страницы.
3. При загрузке страницы браузером происходит построение DOM-модели. Браузер находит код JavaScript и сразу же начинает читать его, подготавливая к запуску. Если в коде найдется ошибка, браузер будет стараться продолжить чтение, чтобы пользователю не отказали в показе запрошенной страницы.
4. Браузер начинает выполнять JavaScript-код в момент обнаружения. При этом нужно помнить, что код выполняется вплоть до закрытия страницы, а не до окончания ее формирования (как это происходит, например, в PHP). Актуальные версии JavaScript имеют высокую производительность. Но надо понимать, что код выполняется на стороне клиента и расходует его ресурсы, а не сервера.

Научимся подключать JavaScript-код к странице. Это делается при помощи HTML-тега **<script>**. Напишем простую страницу. Пока не будем вникать в код JavaScript— просто подключим его и посмотрим, что произойдет:

```
<head>
<script>
setTimeout(reminder, 5000);
function reminder() {
 alert("Так и будешь смотреть на эту скучную страницу?");
}
</script>
</head>
```

При запуске кода в браузере увидим, что через пять секунд браузер отобразит оповещение с

заданным текстом. Это малая часть возможностей языка, с которыми нам предстоит познакомиться. Но этот пример в духе «Hello, World» дает общее представление о том, как происходит подключение и вызов скрипта.

Бывают случаи, когда код выносят в отдельный файл, подключаемый в HTML-коде:

- если кода много;
- если код работает на многих страницах сайта.

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

В качестве значения атрибута **src** указывается путь к файлу, содержащему скрипт. Браузер скачивает и выполняет его. Можно указать скрипт, который расположен на другом сервере:

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.7/angular.min.js"></s
cript>
```

# Структура кода

В JavaScript код состоит из команд. Сама по себе команда описывает операцию. Набор команд, из которых состоит код, и задает поведение страницы. Команды в коде, представленном ниже, объявляют переменные, в которых будут храниться значения. В дальнейшем они повлияют на принятие решений.

```
var car = "Audi";
var power = 500;
if (power < 400) {
 alert("This page is for fast cars only!");
} else {
 alert("Welcome " + car + " owner!");
}
```

В переменных могут храниться числа, строки, логические значения и другие данные — познакомимся с ними позже. Вне зависимости от типа данных все переменные создаются по одному алгоритму.

Чтобы объявить переменную, начинают с ключевого слова **var**. Затем пишут имя переменной. При необходимости можно указать ее начальное значение через знак равенства, но это необязательно, так как его можно присвоить позднее. Переменную можно объявить и без ключевого слова **var**, но в таком случае она сразу становится глобальной. О минусах такого решения поговорим чуть позже.

```
var power = 500;
var speed;
```

Имя переменной должно начинаться с буквы, подчеркивания или знака $. Потом может следовать любое количество букв, цифр, подчеркиваний и знаков $. В JavaScript есть набор ключевых слов, использовать которые в качестве имен переменных не следует. Вот их полный список:

- break;
- case;
- catch;
- class;
- const;
- continue;
- debugger;
- default;
- delete;
- do;
- else;
- enum;
- export;
- extends;
- false;
- finally;
- for;
- function;
- if;
- implements;
- import;
- in;
- instanceof;
- interface;
- let;
- new;
- package;
- private;
- protected;
- public;
- return;
- static;
- super;
- switch;
- this;
- throw;
- true;
- try;
- typeof;
- var;
- void;
- while;
- with;
- yield.

В JavaScript регистр символов имеет значение практически везде. Поэтому будьте внимательны, давая имена переменным.

Когда пишете код и даете переменным имена вроде **_qwe**, **$** или **bar**, помните, что ваш код, возможно, будет читать еще кто-то. И даже если эти наименования понятны и осмыслены лично для вас, коллега их не поймет. А имена **currentSpeed**, **isValid** сразу показывают, что за значение хранится в них.

Предположим, что вы пишете код и надо подобрать имя для переменной, в которой хранится массив с четными строками текста красного цвета. В данном случае удобно использовать стиль **lowerCamelCase** — начинайте с прописной буквы каждое слово, кроме первого: **evenRedStringsText**.

Переменные, имена которых начинаются с **$**, резервируются для библиотек JavaScript и хранения DOM-узлов. Некоторые авторы применяют такие имена, но это не рекомендуется.

Чтобы выполнять действия, в JavaScript применяются выражения. Они вычисляются, после чего становится доступен их результат в виде значения:

```
var distance = time * speed;
var fullname = "Mr " + "Rick " + "Sanches";
power < 400; // логическое выражение
```

Здесь используется комментарий. Он может находиться в любом месте программы. Комментарии не влияют на выполнение кода — интерпретатор игнорирует их. Они используются, чтобы оставить важную информацию об участке кода или временно деактивировать его, не удаляя.

- Однострочные комментарии начинаются с двойного слэша **//**.
- Многострочные — со слэша и звездочки **/*** и заканчиваются звездочкой и слэшем ***/**.

Помимо переменных в JavaScript можно определять символьные имена для конкретных значений. Если есть цвет #FF7F00, каждый раз писать его значение неудобно — можно положить его в константу с удобным именем. Как правило, такое наименование пишут большими буквами через подчеркивание.

```
var COLOR_RED = "#F00";
var COLOR_GREEN = "#0F0";
var COLOR_BLUE = "#00F";
var COLOR_ORANGE = "#FF7F00";
var color = COLOR_ORANGE;
alert( color ); // #FF7F00
```

Чисто технически — это обычная переменная. Но особое написание продиктовано договоренностью не менять значение в процессе написания кода.

# Типы данных

Типы данных в JavaScript делятся на две группы: простые типы и объекты. Простые типы — это числа, текстовые строки и логические, или булевы, значения. Специальные значения **null** и **undefined** — это простые значения, но они не являются ни числами, ни строками, ни логическими значениями. Каждое из них определяет только одно значение собственного специального типа.

Значение, которое не является числом, строкой, логическим значением или специальным значением **null** или **undefined** — это объект, то есть коллекция свойств, каждое из которых имеет имя и значение.

## Тип number (число)

```
var n = 123;
n = 12.345;
```

Нужно учитывать, что тип **number** (число) используется как для целых, так и для дробных чисел. У него есть специальные числовые значения **Infinity** (бесконечность) и **NaN** (ошибка вычислений, **not a number**).

**Infinity** получается при делении на ноль:

```
alert( 1 / 0 ); // Infinity
```

Ошибка вычислений NaN будет результатом некорректной математической операции:

```
alert("не число" * 2 ); // NaN, ошибка
```

## Тип string (строка)

```
var str = "Мама мыла раму";
str = "Одинарные кавычки тоже подойдут";
```

В JavaScript одинарные и двойные кавычки равноправны. Выберите для себя один тип и будьте последовательны.

## Булевый (логический) тип boolean

У данного типа есть два значения: **true** (истина) и **false** (ложь).

```
var checked = true;  // поле формы помечено галочкой
checked = false;     // поле формы без галочки
```

## Специальное значение null

**null** не относится к вышеперечисленным типам. Он указывает на несуществующий объект — это просто специальное значение для понятия «ничто».

```
var age = null;
```

## Специальное значение undefined

Этот особый тип фигурирует, когда переменная объявлена, но значение ей еще не успели присвоить.

```
var x;
alert( x ); // выведет undefined
```

## Тип object (объекты)

Перечисленные выше типы называют примитивными. Тип «объекты» применяется для хранения коллекций данных и объявления сложных сущностей. Подробно объекты изучаются в продвинутом курсе JavaScript.

В любом месте кода можно узнать, какой тип имеет та или иная переменная. Для этого применяется оператор **typeof**. Он возвращает тип переданного аргумента в виде строкового значения.

```
typeof undefined        // undefined
typeof 0                // number
typeof true             // boolean
typeof "foo"            // string
typeof {}               // object
typeof null             // object
typeof function(){}     // function
```

Обратите внимание на поведение последних двух строк:

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 будем обсуждать позже, но сейчас отметим, что они не являются отдельным базовым типом в JavaScript, а представляют подвид объектов. При этом **typeof** выделяет функции, возвращая для них ***function***. Это удобно, так как позволяет легко определить функцию.

# Стандарт языка

Довольно долго язык JavaScript развивался, сохраняя обратную совместимость. Новые возможности добавлялись, а старые не менялись, чтобы не сломать существующие HTML/JS-страницы. Это привело к тому, что любая ошибка в дизайне языка фиксировалась в нем навсегда.

Так продолжалось до появления стандарта **ECMAScript 5 (ES5)**, который добавил новые возможности, а также привнес в язык множество исправлений. И старый код перестал работать.

Чтобы нивелировать эту проблему, разработчики решили, что по умолчанию опасные изменения будут выключены, а интерпретатор языка будет работать по-прежнему. Чтобы перевести код в режим полного соответствия современному стандарту, требуется указать специальную директиву **use strict** (не поддерживается браузерами младше IE9).

Сама директива — это строка ***use strict***, которая ставится в начале скрипта:

```
"use strict";
```

При использовании этой директивы нельзя будет, например, создавать переменные без ключевого слова **var**, что разрешено в более старом стандарте.

# Практическое задание

1. Задать температуру в градусах по Цельсию. Вывести в **alert** соответствующую температуру в градусах по Фаренгейту. Подсказка: расчет идет по формуле **Tf = (9 / 5) * Tc + 32**, где **Tf** — температура по Фаренгейту, **Tc** — по Цельсию.

2. Объявить две переменные: **admin** и **name**. Записать в **name** строку "Василий"; Скопировать значение из **name** в **admin**. Вывести **admin** (должно вывестись «Василий»).

3. * Чему будет равно JS-выражение 1000 + "108"?
4. * Самостоятельно разобраться с атрибутами тега **script** (**async** и **defer**).

# Дополнительные материалы

1. Что такое JavaScript.
2. JavaScript для начинающих — как учить?

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Дэвид Флэнаган. JavaScript. Подробное руководство.
2. Эрик Фримен, Элизабет Робсон. Изучаем программирование на JavaScript.