Введение

Перед тем как мы начнём изучать javascript, давайте вспомним чего нам не хватает в верстке html/css и подумаем, как будет лучше всего этот материал изучить.

Давайте вспомним что такое html?

HTML – стандартный язык разметки документов во Всемирной паутине.

Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на мониторе компьютера или экране мобильного устройства.

Одним из самых частых сравнений технологий с реальной жизнью является сравнение с человеком или с автомобилем, давайте рассмотрим сравнение с автомобилем. Первое что мы создаём это каркас автомобиля, эта часть автомобиля определенно одна из самых важных, но ее явно недостаточно, для того чтобы придать этому “каркасу” автомобиля прекрасный внешний вид нам нужно изучить CSS

Что такое CSS?

CSS – каскадные таблицы стилей – формальный язык описания внешнего вида документа, написанного с использованием языка разметки (HTML).

Получается что css без html абсолютно бесполезен и эти 2 технологии используются вместе, с помощью css мы можем задать любую стилистику для html страницы и как в нашем сравнении мы можем задать любой внешний вид для нашего автомобиля.

Чего же нам еще не хватает, просто если мы такой автомобиль поставим где-то на парковке, он будет смотреться идеально и никто не догадается что у него еще нет двигателя и может быть даже капот открыть нельзя или у него еще нет стеклоподъемников, именно для этого нам и необходимо добавить эту взаимосвязь с пользователем. Для этого и используется Javascript на сайте.

Javascript – язык программирования. С помощью которого мы можем добавить любой интерактив на страницу, любое взаимодействие с пользователем. Простыми словами это язык программирования который может реализовать любой функционал на вашем сайте в зависимости от того, что за действие вы выполняете на сайте, это может быть ввод значений в поле ввода и нажатие на элемент левой клавишей мыши.

JavaScript — одна из основополагающих технологий современного веба и практически его ровесник. За двадцать лет своего развития JavaScript стал достаточно мощным языком программирования. Он остаётся наиболее широко используемым языком программирования по версии Stack Overflow.

Всё началось с добавления в браузер интерпретатора языка, который мог бы исполнять подключённые к странице скрипты, а сегодня движок этого скрипта используется не только в браузере. Если все эти слова для вас кажутся сложными и непонятными, давайте детально во всем разбираться!

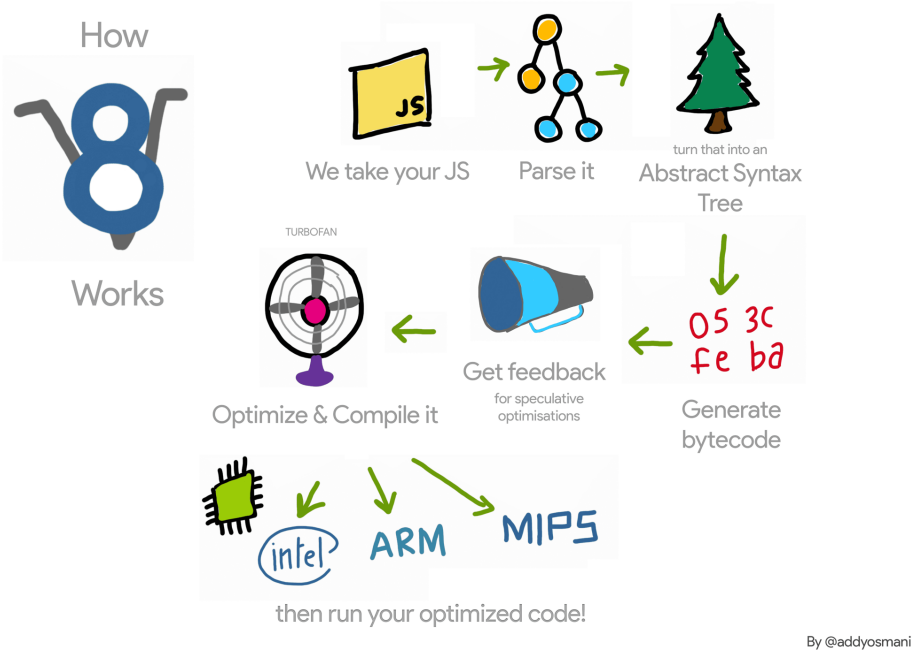
Программы, содержащие движок для выполнения JavaScript. Интерпретатор

Разберемся, каким образом работает код на JavaScript. Чтобы машина поняла и исполнила то, что мы пишем на этом языке, нам нужна специальная программа, которая выполнит наш скрипт. Примеры таких программ:

- веб-браузер;
- среды выполнения Node.js, Deno, Electron;
- любое другое ПО, которое можно установить на робот-пылесос, или визуальная среда управления ракетой Falcon 9.

Все эти программы объединяет наличие **движка** (JavaScript engine). Движок — это компьютерная программа, на практике написанная на C++, хотя можно написать её и на другом языке. Она исполняет код на JavaScript путём чтения и обработки (parsing) скриптов и превращения их в машинный код (интерпретатор).

Наиболее известный движок — V8. Он разработан Google и исполняет JavaScript в браузере Google Chrome. Также есть движок Rhino, разработанный компанией Mozilla Foundation. Он применяется в браузере Firefox.



(источник)

Среды выполнения

Движок для исполнения JavaScript, программа, в которой этот движок работает, и операционная система вместе составляют среду выполнения.

Примеры сред выполнения:

- Windows/MacOs плюс браузер Chrome;
- Linux плюс Node;
- упрощённая версия Nix-системы плюс движок V8.

Java и Javascript одно и тоже?

Ну что, давайте ответим на самое частое заблуждение, JavaScript схож с Java только по названию – их не стоит путать. JavaScript был создан во время одного из пиков популярности Java, что позволило разработчикам языка успешно применить элементы синтаксиса языков семейства C. В целом Java и JavaScript совершенно непохожи.

Стандарты

Первоначально язык зародился без стандартов, но позже был передан в организацию, занимающуюся разработкой стандартов, Ecma International. Именно там были выработаны первые и все последующие стандарты языка, которые включают все тонкости того, как должен

быть реализован интерпретатор, какой функционал будет у языка, какие ключевые слова и поведение.

Так как компания по стандартизации называется Ecma, а слово Java в названии языка JavaScript сбивало с толку, потому что есть полноценный язык программирования Java, стандарт для JavaScript стали называть ECMAScript. В 2015 году вышел релиз ECMAScript, который существенно изменил синтаксис языка. Вы можете столкнуться в сети с программами и примерами кода, которые используют предыдущий стандарт ES5. На нём писали и после 2015 года, и в настоящее время программы поддерживают его. Но абсолютное большинство новых программ пишут на современном синтаксисе ES6+. В этом курсе практически все примеры будут использовать современный стандарт ES6+ (ES2020 на момент создания курса).

Принципы написания кода на JavaScript

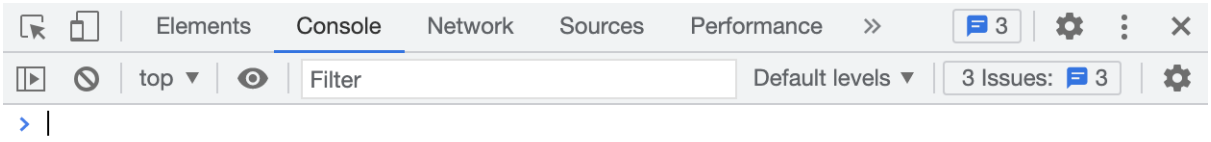
Если Вы уже знакомы с некоторыми языками программирования, то точно знаете, что программа появляется в процессе вполне определенных шагов. Сначала пишется код, при необходимости он компилируется, собирается в пакет, устанавливается на компьютер. Язык JavaScript несколько отличается по своему циклу разработки от данной схемы. Разработчик подключает код JavaScript к странице, которая загружается в браузер. После этого уже сам браузер совершает всё необходимое, чтобы выполнить код.

1. Страница создаётся по стандартной схеме. Верстается структура HTML, к ней применяются стили, в неё добавляется контент сайта.
2. К странице подключается код JavaScript. Как это сделать мы рассмотрим чуть позже
3. При загрузке страницы браузером происходит построение DOM-модели(что такое DOM-модел мы подробно обсудим на 6 уроке, пока это просто тот код, который вы могли видеть в отладчике браузера). Браузер находит код JavaScript и сразу же начинает читать его, подготавливая к запуску. Если в коде будет найдена ошибка, браузер будет старается продолжить чтение, избегая отказа от показа пользователю 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 её формирования, как это происходит, например, в PHP). Актуальные версии JavaScript имеют высокую производительность. Однако, надо понимать, что код выполняется на стороне клиента и расходует именно клиентские ресурсы, а не ресурсы сервера.

Какой вывод мы можем сделать, что подключить script вы можете и внутри тега <head> и внутри <body> в верхней части или перед закрывающимся тегом </body> но на начальном этапе будет самым оптимальным решением выбрать в нижней части сайта перед </body>. Нам необходимо чтобы на странице были все элементы, а потом уже с ними можно было работать с javascript.

Консоль браузера, первые шаги

Очень удобной особенностью работы с языком программирования javascript является то, что вам не обязательно устанавливать редактор кода, чтобы написать свою первую строку кода, для это вам нужно открыть знакомый вам отладчик браузера и перейти во вкладку console



Именно здесь вы можете протестировать создание переменных или вывести “Hello world”, но этот инструмент нам еще сильно поможет при поиске ошибок или отладки кода, вторая часть нам нужна уже сейчас, как быть в ситуации, когда нужно вывести данные переменных или работы функции для проверки, но пользователь не должен их видеть?

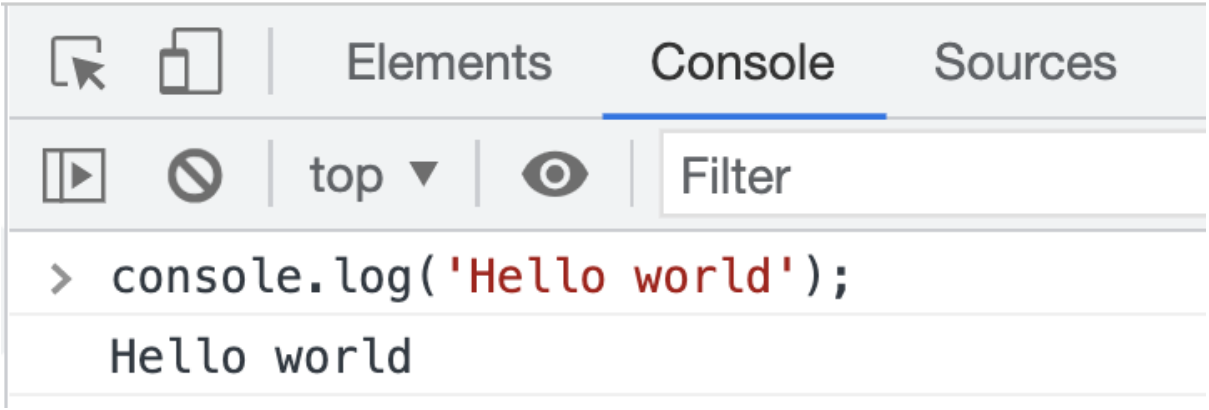
console.log

Это первый и самый удобный способ который мы рассмотрим. Как только вы открыли отладчик браузера и выбрали раздел console, уже здесь вы можете написать console.log(‘Hello world’), как только вы нажмете enter вы увидите результат работы кода

Именно здесь вы можете протестировать создание переменных или вывести “Hello world”, но этот инструмент нам еще сильно поможет при поиске ошибок или отладки кода, вторая часть нам нужна уже сейчас, как быть в ситуации, когда нужно вывести данные переменных или работы функции для проверки, но пользователь не должен их видеть?

console.log

Это первый и самый удобный способ который мы рассмотрим. Как только вы открыли отладчик браузера и выбрали раздел console, уже здесь вы можете написать console.log(‘Hello world’), как только вы нажмете enter вы увидите результат работы кода



Переменные, область видимости, поднятие переменных

Переменные в программировании нужны, чтобы хранить информацию и при необходимости её читать, изменять, сравнивать и так далее. Переменные можно создать с помощью ключевых слов **let**, **const** и **var**.

var — от английского variable («переменная») — устаревший способ объявления переменных, который имеет пару недостатков (мы поговорим о них чуть позже) и не рекомендуется к использованию в современном коде.

let и **const** (let с английского — «пусть будет», а const — «константа») — это два новых ключевых слова, которые были добавлены в язык со стандартом ES2015. Они лишены недостатков, присущих var.

При объявлении переменной мы можем сразу же задать ей значение, это называется инициализацией переменной.

Начнём изучать способы объявления переменных с var.

var

Ключевое слово **var** относится к стандарту ES5. Написать в программе var не будет ошибкой, однако, если мы пишем программы на ES6 в современных реалиях, мы используем только let и const. Два ключевых момента при использовании var:

1.

```
var notInitialized; // неинициализированная переменная.
var age = 20; // инициализированная значением 20 переменная.
var age = 10; // нет ошибки несмотря на то, что мы использовали уже занятое
имя, тем самым переопределив значение в другой переменной, которая была
создана ранее. В старом коде можно было легко создать переменную с тем же
именем, что уже есть в другом скрипте на странице, тем самым предопределив её
значение и сломав чужой код.
```

Создание переменной с тем же именем через let или const вызовет ошибку, так как в новом стандарте специально убрали такую возможность:

```
let age = 20;
let age = 10; // SyntaxError: Identifier 'age' has already been declared — с
использованием let мы не можем переопределить чужую переменную, и всё будет
работать хорошо.
```

2. Всплытие (hoisting) или поднятие переменной.

Всплытие переменной — это особенность языка JavaScript, которая позволяет обратиться к значению переменной до её объявления через var. При этом в значении будет undefined и не будет ошибки обращения, как в случае использования переменной, объявленной через let. Это может приводить к ошибкам в коде, которые сложно обнаружить.

```
console.log(age); // undefined
var brotherAge = age + 12;
console.log(brotherAge); // NaN
var age = 10;

console.log(name); //
name = "Ivan"; // Uncaught ReferenceError: name is not defined
```

let

Новый способ объявления переменных, который позволяет создавать переменные с блочной областью видимости. Например, если такую переменную создать внутри условного блока, то её значение будет доступно только внутри этого блока. (область видимости мы разберём на уроке про функции)

```
let result = 8;
console.log(result); // 8

let updatedResult = result + 2;
console.log(updatedResult); // 10
```

const

Если переменная объявлена через const, её значение изменить нельзя. Правильнее называть такие значения константами.

```
const GAP_SIZE = 5;
GAP_SIZE = 3; // TypeError: Assignment to constant variable.
```

Имена переменных

Имена переменных чувствительны к регистру.

```
let result = 5;
let Result= 3;
console.log(result); // 5
console.log(Result); // 3
```

result и Result — это разные переменные. Если вы пишете программу, помните, что имена переменных предназначены в первую очередь человеку, их читающему, — другому разработчику или себе в будущем, поэтому они должны быть легко читаемы и легко отличимы, а также рассказывать своим именем, что в них хранится.

Типы данных

В JS выделяют 8 типов данных:

- строка (**`string`**);
- число (**`number`**);
- булево значение (**`boolean`**);
- **`undefined`**;
- объект (**`object`**);

- `null`;
- `symbol`;
- большое число (`BigInt`).

Чтобы определить, к какому типу данных относится значение, мы можем применить оператор `typeof`:

```
console.log(typeof 2) // "number"
console.log(typeof "2") // "string"
```

Такое количество типов позволяет работать с разными данными и выбирать подходящие типы для их хранения. Например, для хранения имён или разных предложений нам подойдёт строковый тип, а когда нам нужно производить вычисление среднего балла или другие математические действия, для хранения данных подойдёт числовой тип. Для хранения сложных структур, например адреса, состоящего из нескольких полей, хорошо подходят объекты, а для хранения большого списка однородных элементов, например ключевых слов для статьи, прекрасно подходят массивы.

В чём отличия данных разных типов:

- по-разному сравниваются;
- имеют разные встроенные методы для работы со значениями.

Строка

Тип данных строка — это текст, заключенный в кавычки. Двойные и одинарные кавычки не имеют различий. Кроме того, строка может быть заключена в обратные кавычки (backticks `). Они позволяют передавать в строку переменную или вызов функции. Такие строки называются строковыми шаблонами.

```
let string = "Hello";
string = 'Hello';

let result = 8;
let literal = `Результат: ${result}`; // Результат: 8
let emptyString = "";
```

Для работы со строками в языке есть много вспомогательных методов и свойств. Например, можно получить длину строки, объединить две строки вместе или узнать, является ли одна

строка подстрокой второй строки. Полное описание методов и свойств строк можно найти в MDN.

Число

Тип данных число — это число, целое или десятичное. Также сюда относятся числа в двоичной, восьмеричной и шестнадцатеричной системах счисления. В JavaScript для типа `number` установлено ограничение на максимальные значения от $-(2^{53}-1)$ до $(2^{53}-1)$.

В языке есть несколько специальных значений, которые определяются как число. То есть при вызове `typeof` мы для них получим `"number"`:

- NaN;
- Infinity, -Infinity.

NaN возникает при ошибке вычисления и «поглощает» все другие арифметические операции, которые мы пытаемся производить над переменной. Возникает, когда движок не может получить результат математической операции, например когда один из операторов не число.

```
let numberGradeValue = 5;
let stringGradeValue = "A";
const result = numberGradeValue - stringGradeValue; // NaN
```

Infinity — бесконечность. Часто получается в результате деления на 0. При попытке деления на 0 программа не зависает и не бросает ошибок, а просто присваивает результату значение `Infinity`. Как и с NaN, с этим значением арифметически ничего не сделать. Его очень удобно использовать для задания бесконечных значений вместо очень больших чисел. Можно в разных циклах или условиях проверять на `Infinity`. Например, при поиске минимального значения нужно первоначально задать переменной наибольшее значение, для этого отлично подойдёт `Infinity`.

-Infinity — это отрицательная бесконечность. Например, от деления отрицательного числа на 0.

Для работы с числами в JavaScript есть множество встроенных полезных функций, которые вы всегда сможете найти в MDN.

Булево значение

Булево (логическое) значение может быть только правдивым (**`true`**) или ложным (**`false`**). Оно применяется в тех местах, где нам нужно именно логическое значение, например включено/выключено, содержит/не содержит и т. п. Чаще всего булевы значения используются как некие флаги для указания состояния и служат для задания логики программы, проверки значения на правду (**`truthy`**) или ложь (**`falsy`**).

```
let isPassExam = true;
if (isPassExam) console.log('Поздравляем, Вы сдали экзамен!'); // Поздравляем, вы сдали экзамен!
```

Undefined

`Undefined` — это специализированный тип данных, который имеет одно-единственное значение **`undefined`** и автоматически присваивается вновь созданным, но не инициализированным другим значением переменным. Также, если в функции не определён возврат какого-либо значения, она возвращает **`undefined`**.

Этот значение позволяет проверить, инициализировали ли мы переменную.

```
let isPassExam;
console.log(isPassExam); // undefined
if (isPassExam) console.log('Поздравляем, Вы сдали экзамен!'); // '', так как undefined в условиях приводится к false (смотри ниже про приведение типов).
```

Объект

Предыдущие типы данных были примитивными. Сейчас мы рассмотрим тип, который от них отличается.

Часто нам необходимо хранить не одно примитивное значение, а набор данных (набор примитивных значений). К примеру, у нас есть карточка студента, которую мы хотим представить в программе. Мы можем создать объект `student` и перечислить в нём все необходимые данные. Это будет удобнее, чем создавать несколько примитивных переменных для каждого типа данных.

Обращаться к полям (свойствам) объекта можно через знак “.” (`student.firstName`) или указывая имя ключа (свойства) в квадратных скобках (`student['lastName']`).

```
let student = {
  firstName: "Ivan",
  lastName: "Petrov",
  age: 33,
  faculty: "Information Technologies"
};

console.log(`${student.firstName} ${student.lastName} is ${student.age} years
old`); // Ivan Petrov is 33 years old.
```

В JavaScript объекты могут быть созданы пользователем, а также есть два типа данных, которые тоже являются объектами, но имеют конкретное предназначение и используются самостоятельно — это **Array (массив)** и **Function (функция)**.

Array

Массив позволяет хранить неограниченное количество любых данных как упорядоченный список элементов. У каждого элемента есть свой индекс, который начинается с 0. Также массив имеет **длину** (`length`) — это количество элементов (ячеек) в массиве. Обращение к элементам массива идёт по индексу, указанному в квадратных скобках после имени массива. Пример массива:

```
let listOfBooks = ['Стив Макконел "Совершенный код"', 'Роберт Мартин "Чистый
код"'];
console.log(listOfBooks[0]); // Стив Макконнелл "Совершенный код"
console.log(listOfBooks.length); // 2 - длина массива (2 элемента).
```

Function

Функция в JavaScript — это объект, она имеет свои свойства и методы, может быть сохранена в переменную, передана как аргумент в другую функцию. Именно это делает язык JavaScript очень мощным и позволяет работать с такими вещами, как **замыкания** и **функциональное программирование**.

У функции, как и у других типов данных, есть специализированные методы и свойства. Один из самых используемых методов функций — это `toString()`, который возвращает строковое представление функции. Чаще всего это текст кода функции, но он может быть переопределён для получения особых результатов.

Полный список свойств и методов функции можно посмотреть в MDN.

Пример функции:

```
const add2= function(x) {
  return x + 2;
}

console.log(add2.toString()); // function(x) { return x + 2; }
console.log(add2(10)); // 12 - выводится результат выполнения функции.
```

Null

`Null` — это специальное значение, которое используется с переменными объектного типа для указания, что в переменной нет объекта. Сам **`Null`** является объектом, но при этом никуда не ссылается. **`Null`** часто используют в API, когда в переменной ожидается объект, но по каким-то причинам API не может найти подходящего объекта и возвращает **`Null`**.

`Null` — примитивный тип. В операциях сравнения и булевой логике он выступает как ложное (**`falsy`**) значение.

Symbol

Это новый тип данных, который был создан, чтобы создавать уникальные ключи для свойств объектов или каких-либо других целей. Дело в том, что ключи объекта в виде строки могут повториться в вашем коде или совпасть с ключами объектов в сторонних библиотеках, работающих параллельно с вашим кодом. Чтобы избежать такой ситуации, в ECMAScript 2015 добавили новый тип данных **`Symbol`**.

Для создания переменной такого типа необходимо вызвать функцию (**конструктор**) этого типа **`Symbol()`**. Также в качестве аргумента в эту функцию можно передать строку, что позволит создать символ на основе вашей строки. Создав два символа на основе одинаковой строки, вы всё равно получите два разных значения. Символ — это не строка, поэтому пытаться вывести его напрямую не стоит. У него, как и у функции, есть метод **`toString()`**. Посмотрим на примере:

```
const uniqKey= Symbol();
console.log(uniqKey.toString()); // Symbol()
const uniqKey2 = Symbol('test');
const uniqKey3 = Symbol('test');
console.log(uniqKey2.toString()); // Symbol('test')
console.log(uniqKey2 == uniqKey3) // false - символы всегда создаются
уникальными.
```

BigInt

Когда стандартных чисел типа `number` мало — при сложных расчётах или при использовании меток времени с микросекундами, — можно использовать супербольшие значения чисел (только целые числа).

Такие числа записываются как обычное целое число, только в конце добавляется литера n.

```
let bigNumber = 63374851375010000n;
console.log(bigNumber); // 63374851375010000n
```

Дополнительные материалы

1. Разбираемся с поднятием (hoisting) в JavaScript — статья на medium.com.
2. Вы не знаете JS: типы данных и значения — статья на medium.com.
3. Преобразования типов.
4. Строка.
5. Число.
6. Что за чёрт, Javascript — статья на Хабре, раскрывающая множество так называемых странностей JavaScript. Может быть полезно прочитать перед собеседованием.

Используемые источники

1. Вы не знаете JS: типы данных и значения — статья на medium.com.
2. Документация MDN.