

Введение в JavaScript

JavaScript это...

- Интерпретируемый
- Сценарный
- Объектно-ориентированный
- Исполняется в контексте
- Синтаксис похож на C, C++, Java
- Динамическая типизация. Автоматическое преобразование типов
- Отсутствуют классы, а наследование объектов реализовано через прототипы
- Функции выступают как объекты

JavaScript Победил!

- Создание веб-приложений
 - Серверная часть (NodeJS, RingoJS)
 - Клиентская часть (DOM, AJAX, Comet)
- Букмарклеты
- Встраиваемые платформы
- Мобильные приложения (в браузере мобильного устройства, PhoneGap, Appcelerator Titanium)
- Виджеты и расширения для браузеров
- Прикладное программное обеспечение(ПО для Windows 8)
- Манипуляция объектами приложений (продукты Adobe, MS Office)

Повлиял

- Objective-J
- Dart
- TypeScript
- CoffeeScript
- Haxe
- JSON
- ActionScript

Запуск скрипта

- JavaScript всегда выполняется в рамках глобального объекта. В браузере это **window**, в Node JS – **global**.
- Из командной строки (Rhino, V8, NodeJS, RingoJS)

```
//Run with Rhino  
java -jar js.jar -f myapp.js  
//Run with NodeJS  
node myapp.js
```

- In Browser
 - HTML тег <script> с атрибутом type="text/javascript"
 - Адресная строка со спецификатором: javascript
 - С помощью Dev Tools (Google Chrome Web Inspector, FireBug, Средства разработчика в IE)

Переменные

- Область видимости(scope): глобальная (window или global) и локальная (функции, with, catch)
- Отсутствует блочная область видимости
- Объявляются с помощью **var**.
- Если опустить это слово, то переменная станет свойством глобального объекта
- Обращение к необъявленной переменной вызовет ошибку
- Все переменные объявленные в одной области видимости «всплывают» в начало
- Можно присваивать значения различных типов
- Нельзя явно удалить (в отличие от свойств объектов, которые можно удалить с помощью оператора delete). Можно только подсказать сборщику мусора, что переменная больше не нужна задав значение null.

```
var age = 25,  
    name = "Миша";  
console.log(name); // Миша  
console.log(isJS); // undefined  
var isJS = true;  
console.log(isJS); //true  
age = "Задали другое значение. В JS это можно.";
```

Переменные

```
var myVar = 3; // Объявили переменную
myVar = "This is String!"; // Изменили тип
// Все объявления переменных всплывают автоматически в рамка своей
// области видимости, поэтому лучше объявить их сразу в начале функции!
var n = 3,
s = "My String";
n = 4;
s = n + " " + s;
var t = n * n;
var n = 43;
// эквивалентно следующему:
var n, s, t; // они undefined!
n = 3;
s = "My String";
n = 4;
s = n + " " + s;
t = n * n;
n = 43;
```

Идентификаторы

```
var age = 25,  
    name = "Миша";  
console.log("Я " + name + " и мне " + age + " лет.");
```

- Могут содержать: \$, _, a-Z, 0-9
- Не могут начинаться с цифры
- Не могут совпадать с зарезервированными словами

```
abcd = true;  
_ = true;  
$ = true;  
var = false;  
_$$$$$$$$$_ = true;  
100$ = false;  
$200 = true;  
имя_пользователя = false;
```



Зарезервированные слова

- **Используются в JS**

break; do; if; switch; typeof; case; else; in; this; var;
catch; false; instanceof; throw; void; continue; finally;
new; true; while; default; for; null; try; with; delete;
function; return

- **Не используются в JS**

abstract; double; goto; native; static; boolean; enum;
implements; package; super; byte; export; import;
private; synchronized; char; extends; int; protected;
throws; class; final; interface; public; transient; const;
float; long; short; volatile; debugger

Типы. Number

- Нет различий между вещественным числом и целым
- Передается и присваиваются по значению
- Можно сравнивать, складывать, отнимать, делить, умножать, получать остаток от деления(%)
- Максимально-возможное значение можно определить: Number.MAX_VALUE
- Все что больше/меньше максимального/минимального (+-)Infinity
- NaN (Not A Number)

```
// Литеральный способ
```

```
var year = 2013,  
    price = 3.99;  
console.log(year + price - 10); // 2006.99
```


```
//С помощью конструктора
```

```
var nextYear = new Number(2014); // Bad practice! typeof=>object!
```

Типы. Number

```
var isNumber = function (n) {  
    return !! (toString.call(n) === "[object Number]");  
}
```

```
console.log(typeof 3); //=> number  
console.log(typeof new Number(3)); //=> object!  
console.log(3 === 3); //=> true  
console.log(new Number(3) === 3); //=> false!  
console.log(Number(3) === 3); //=> true  
console.log(new Number(3) === new Number(3)); //=> false!
```



Типы. String

- Строковые литералы указываются в “ или ‘
- Передаются и присваиваются по значению
- Для экранирования используется обратный слеш
- Значение должно записываться в одной строке
- Строку невозможно изменить
- Длину строки можно определить с помощью свойства length
- Можно использовать управляющие последовательности(\n, \t и т.д).
- Объединять строки можно с помощью «+»
- Обращаться к символам можно по их индексам

```
// Литеральный способ
var lang = "JavaScript";
    console.log('I love ' + lang); // I love JavaScript
//С помощью конструктора
var name = new String("Denis"); // Bad practice! typeof=>object!
```

Типы. String

```
var isString = function (n) {  
    return !! (toString.call(n) === "[object String]");  
}
```

```
console.log(typeof "hello"); //=> string  
console.log(typeof new String("world")); //=> object!  
console.log("Hello" == "hello"); //=> false!  
console.log("Hello" === new String("Hello")); //=> false!  
console.log("JavaScript".length); //=> 10  
console.log("*****  
        ***JS***  
        *****"); //=> Error!
```

```
var name = "abcdef";  
console.log(name[1]); //=> "b"  
name[4] = "z";  
console.log(name); //=> "abcdef"
```



Типы. Boolean

- Могут принимать только 2 значения: **true, false**
- Поменять значение на противоположное можно с помощью: **"!"**

```
// Литеральный способ
var isMan = true,
    isWoman = !isMan;
console.log(isMan, isWoman); //true, false
//С помощью конструктора
var isSuperMan = new Boolean(true); // Bad practice!
//typeof=>object!
```

Типы. Boolean

```
var isBoolean = function (n) {  
    return !! (toString.call(n) === "[object Boolean]");  
}
```

```
console.log(typeof true); //=> boolean  
console.log(typeof new Boolean(false)); //=> object!  
console.log(!true); //=> false  
console.log (!!true); //=> true  
console.log (!!new Boolean(false)); //=> true!
```



Преобразование типов

```
// To Number
// Автоматическое преобразование: Сделать мат. действия (умножить, отнять, разделить)
// Явное
var price = Number("45");
// или использовать методы parseInt, parseFloat.
var size = parseInt("200px", 10); //=> size === 200.

// To String
// Автоматическое преобразование: сложить со строкой.
// Обратите внимание:
var price = 3 + "99" //=> "399", а не 102!
// Явное
var price = String(3.99);
// or
var subPrice = 3.99.toString();

// To Boolean
// Автоматическое преобразование происходит при проверке условий
// Явное:
var isMan = !!price; // 0, пустая строка, null, undefined - false, остальное - true
// OR
var isMan = Boolean(man);
```


Типы. Null. Undefined

- Null – специально присвоили «НИЧЕГО».
- Undefined - возвращается при обращении либо к переменной, которая была объявлена, но которой никогда не присваивалось значение, либо к свойству объекта, которое не существует.

Типы. Null. Undefined

```
var isNull = function (n) {  
    return !! (n === null);  
}  
var isUndefined = function (n) {  
    return !! (void 0);  
}
```

```
var notDefVar,  
    defVar = "abc",  
    nullVar = null;  
console.log(typeof nullVar); //=> object!  
console.log(typeof notDefVar); //=> undefined  
console.log(notDefVar); //=> undefined  
console.log(defVar[3]); //=> undefined  
console.log(nullVar); //=> null
```



Типы. Object

- Объект – это коллекция именованных значений (свойств)
- Чтобы сослаться на свойство объекта, надо указать имя объекта, затем точку и имя свойства
- Свойства объектов во многом похожи на JavaScript переменные – они могут содержать любой тип данных, включая массивы, функции и другие объекты
- Доступ к свойствам объекта из методов осуществляется с помощью ключевого слова **this**
- Объекты в JavaScript могут выступать в качестве ассоциативных массивов
- Передаются в функции по ссылке
- Все объекты наследуются от базового конструктора **Object**

Типы. Object

//Литерал

```
var order = {  
    price: 3.99,  
    user: null,  
    add: function(val){ // Метод  
        this.price = this.price + val; // Обращение через this  
    },  
    goods: []  
};
```

// Через функцию конструктор с использованием ключевого слова new:

```
var order = new Object();  
order.user = {};  
order.user.name = "Ivan";
```

// Общий случай

```
var order = new Order(3.99); // До этого объявили собственную  
функцию – конструктор Order. Об этом позже
```

Типы. Object

```
var isObject = function (n) {  
    return !! (n === Object(n));  
}
```

```
var order = {},  
    secondOrder = order;  
console.log(typeof order); //=> object  
console.log(order.user); //=> undefined  
order.user = "Ivan";  
console.log(secondOrder.user); //=> Ivan  
secondOrder.user = "Petr";  
console.log(order.user); //=> Petr  
order["_add"] = function() { //Добавили к объекту метод  
    console.log(this);  
};  
order._add(); // order;  
console.log(order === secondOrder); // true
```



Типы. Array

- Массив - это объект
- Обращаться к элементам можно по порядковым номерам
- Не существует многомерных массивов. За-то можно объявлять массивы как элементы
- Элементы массива не обязательно должны быть одного типа
- Изменить/Получить длину массива можно установив/получив значение свойства **length**
- Длина массива не фиксирована
- Оператор **delete** удаляет значение, а не элемент

// Литеральный способ

```
var goods = ["TV", "Camera", "PlayStation"];
```

```
var orders = []; // Пустой массив
```

// С помощью конструктора. Bad practice!

// Если аргумент 1 и это число, то это длина массива.

```
var nums = new Array(3); // nums.length => 3. nums[1] => undefined;
```

// Создаст массив длиной 4. И соответствующими элементами массива.

```
var nums = new Array(3, 0, 10, 100);
```

Типы. Array

```
var isArray = function (n) {  
    return !! (toString.call(n) === "[object Array]");  
}
```

```
var myArr = [10, "is String", null, false, -0.1];  
console.log(typeof myArr); //=> object  
console.log(myArr.length); //=> 5  
console.log(myArr[1]); //=> "is String"  
myArr[23] = ["Google", "Yahooo!", "Bing!"];  
console.log(myArr.length); //=> 24  
console.log(myArr[23][1]); //=> "Yahoo!"  
myArr[15] = function(a) { //Элемент массива - функция  
    console.log(a);  
};  
myArr[15]("hi"); // "hi";  
myArr[2] = {};  
myArr.length = 3; //=> [10, "is String", {}];
```



Типы. Function

- Функция - это объект (Присваивать их переменным. Добавлять собственные свойства и методы. Возвращать из других функций. Передавать в качестве аргументов.)
- Выполняется в рамках объекта (**this**). Если объект задан не явно, то используется глобальный
- Вызвать функцию в контексте другого объекта можно с помощью методов **call**, **apply**.
- Может быть анонимной
- Выступают в качестве конструкторов для объектов
- Позволяет создать локальную область видимости
- Могут принимать любое количество аргументов
- К аргументам можно обратиться с помощью локального объекта `arguments` (похож на массив, но имеет ряд доп. свойств)

// Литеральный способ

```
function calc (arg1, arg2) {  
    return arg1 * arg2;  
}
```

// или

```
var calc = function () {  
    return arguments[0] * arguments[1];  
}
```

// С помощью конструктора. Bad practice.

// Используется только для вычислений на лету

```
var calc = new Function("arg1, arg2", "return arg1 * arg2");
```


Типы. Function

```
var isFunction = function (n) {  
    return !! (typeof n === "function");  
}
```

```
var order = {},  
    getThis = function(){  
        return this;  
    };  
getThis(); // => window/global  
order.get = getThis;  
order.get(); //=> order  
getThis.call(order); // => order  
getThis.each = function(arr, callback){  
    for (var i = 0, count = arr.length; i<count; i++) {  
        callback(i, arr[i]);  
    }  
};  
order.get.each(["first", "second", "last"], function(i, val){  
    console.log("\nItem: " + i + ". Value: " + val);  
}); //=>  
Item:0. Value: first  
Item:1. Value: second  
Item:2. Value: last
```



Типы. Constructors

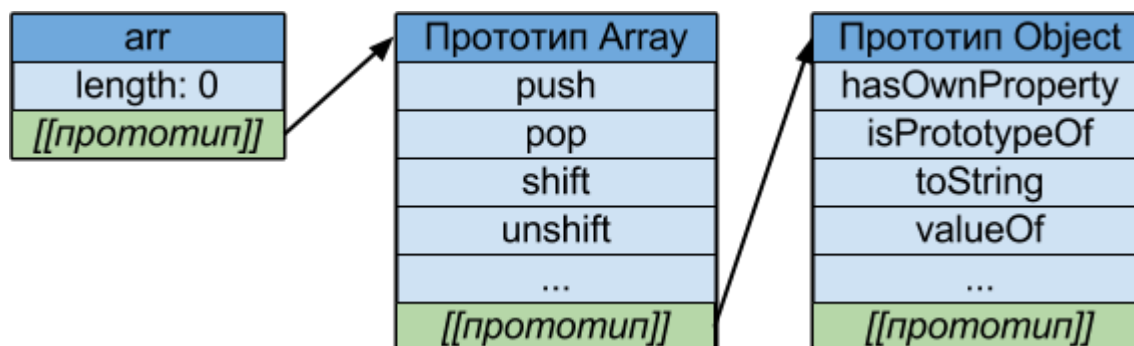
- Конструкторы – это функции которые создают объекты.
- Нет классов в обычном понимании. Наследование реализуется через свойство-объект **prototype**.
- Общее для каждого экземпляра нужно добавлять в прототип
- Все свойства и методы объекта - **public**
- Статические свойства или методы добавляются к конструктору
- Получить ссылку на функцию конструктор объекта можно с помощью свойства **constructor**
- Определить принадлежность объекта к конструктору можно с помощью **instanceof**
- Чтобы создать новый объект вызывайте функцию с оператором **new**
- Объекты можно изменять в любой момент времени
- Предпочтение отдавайте приему составления объектов, а не наследованию

//Как работает прототип

```
var arr = []; // Массивы наследуются от Object.
```

```
arr.push("First Element");
```

```
arr.toString();
```



Типы. Constructors

```
// Создаем конструктор
var Car = function (brand, sn) {
  // Определяем свойства для конкретного экземпляра
  this.brand = brand;
  this.sn = sn;
}
// Добавляем метод в прототип, так как он будет общий для всех экземпляров
Car.prototype.beep = function () {
  console.log(this.brand + ": Beep!");
}
var toyota = new Car("Toyota", "L87WX459087T3"); // Создали новый объект
toyota.beep(); // Toyota: Beep!

var lada = new Car("Lada", "A12OP593K87TZC");
// Переопределим метод для конкретного автомобиля
lada.beep = function () { console.log("Does not work"); };
lada.beep(); // Does not work
toyota.beep(); // Toyota: Beep!

console.log(toyota instanceof Car); // true

var vwPolo = Car("VW", "B01MNRT4123Y5"); // Attention! this === window. Вызвали без new.
```



Типы. Другое

- Date
- RegExp
- Error
- Math
- Arguments и др.

Область видимости. **Scope**

- Глобальная область видимости доступна из любого места программы
- Локальную можно задать только с помощью функций
- Аргументы функций выступают в качестве локальных переменных
- Локальная область видимости доступна для всех вложенных функций

Область видимости. Scope

```
var name = "This is global!";
function localFunction () {
    var localName = "Local Name";
    name = "This is local"; // Переопределили глобальную так как
                           // пропустили var.
}
console.log(name); // "This is global";
localFunction();
console.log(name); // "This is local";
console.log(localName); // Ошибка. Переменная не определена в глобальной области
                        // видимости.
```

```
var a = 3, b = 6, t = 10;
function calc (a, b) {
    var c = 10;
    function plus () {
        var c = 2;
        return (a + b) * c - t;
    }
    return plus();
}
calc(1,2); // -4.
plus(1,2); // Ошибка. Функция не определена, так как она объявлена в локальной
           // области видимости функции calc!
```

Область видимости. Scope

```
// Замыкание:
var uid = (function(){
    var id = 0;
    return function () {
        return ++id;
    }
})();
uid(); // 1;
uid(); // 2;
uid(); // 3;
console.log(id); // Ошибка. Переменная id не определена

// Каррирование:
function curry(x){
    return function(y){
        return x + y;
    }
}
curry(4)(5); // вернёт 9
```

The basic construction. If

Синтаксис:

- **if** (условие) {} **else** {}
- (условие)? вернет если истина : вернет если ложь ;
- Использовать **&&** или **||**

Сложные условия:

- **&&** - логическое И
- **||** - логическое ИЛИ
- **!** - отрицание

Оператор равенства:

- **==** простое сравнение, например:
0 == "" //=> true;
123 == "123" //=> true;
- **===** строгое сравнение, например:
0 === "" //=> false;
123 === "123" //=> false;

The basic construction. If

```
var a = 3, b = 5;
```

```
if (a > b) {  
  console.log(a + "больше " + b);  
} else if (a < b) {  
  console.log(b + "больше " + a);  
} else {  
  console.log("равные");  
}
```

```
function calc (arg1, arg2) {  
  arg2 = arg2 || 10; // 10 - значение по умолчанию  
  return arg1 + arg2;  
}  
calc(3, 9); // 12  
calc(3); // 13  
calc(3, 0); // 13 // Или
```

```
(function(root){ // Создали замыкание  
  var APP = root.APP? root.APP : {}; // Проверяем, если не существует то создаем  
  APP.propertyOne = true;  
  root.APP = APP;  
})(window));
```

The basic construction. Switch

```
switch(переменная) {  
    case 1: // Выполняется, если переменная == 1  
        // Исполняем блок кода 1.  
        break; // Здесь останавливаемся  
    case 2: // Выполняется, если переменная == 2  
        // Исполняем блок кода 2.  
        break; // Здесь останавливаемся  
    case 3: // Выполняется, если n == 3  
        // Исполняем блок кода 3.  
        break; // Здесь останавливаемся  
    default: // Если все остальное не подходит...  
        // Исполняем блок кода 4.  
        break; // Здесь останавливаемся  
}
```

The basic construction. For, While

Можно использовать: **break**, **continue**.

```
while (условие) {  
    // Тело цикла  
}
```

```
do {  
    // Тело цикла  
} while ( условие );
```

```
for (инициализация; условие; инкремент) {  
    // Тело цикла  
}
```

```
for (var переменная in объект) {  
    // тут можно обращаться к свойствам объекта через переменную.  
    Нужно использовать синтаксис ассоциативных массивов. Порядок  
    обхода свойств не определен.  
}
```

The basic construction. For, While

```
for (var i = 1; i < 10; i++) {  
    console.log(i); // 1, 2 ... 9  
}
```

```
var i = 9;  
while (i--) {  
    console.log(i); // 9, 8, 7...1  
}
```

```
console.log('Свойства объекта window');  
for (var name in window) {  
    console.log(name + ":" + window[name]);  
}
```

Обработка ошибок

```
try {  
    /* Может генерироваться исключение либо непосредственно с  
    помощью инструкции throw, либо косвенно. */  
} catch (e) {  
    /* Этот блок может либо каким-либо образом обработать  
    исключение, либо проигнорировать его, делая что-то другое, либо  
    заново сгенерировать исключение с помощью инструкции throw.  
    Переменная e объект типа Error. */  
} finally {  
    /* Этот блок содержит инструкции, которые выполняются  
    всегда, независимо от того, что произошло в блоке try */  
}
```

Литература

- Дэвид Флэнаган, **JavaScript. Подробное руководство**
- Дуглас Крокфорд, **JavaScript: сильные стороны**
- Николас Закас, **JavaScript. Оптимизация производительности**
- Стефанов С., **JavaScript. Шаблоны**
- Алекс Маккоу, **Веб-приложения на JavaScript**