event.propertyName == "value" && calculate(); замечательная штука

{0:1}[0] == 1 вызов объекта первый элемент

При арифметических операциях undefined становится NaN,

Числа

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

NaN (ошибка вычислений). alert( "нечисло" \* 2 ); // NaN, ошибка

**undefined** — «значение не присвоено».

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».

Есть 5 «примитивных» типов: number, string, boolean, null, undefined и 6-й тип — объекты object.

 *Операнд* — то, к чему применяется оператор. Например: 5 \* 2 — оператор умножения с левым и правым операндами. Другое название: «аргумент оператора».

 *Унарным* называется оператор, который применяется к одному выражению. Например, оператор унарный минус "-" меняет знак числа на противоположный:

|  |
| --- |
| var x = 1; |
| alert( -x );     // -1, унарный минус | |

|  |  |
| --- | --- |
| alert( -(x+2) ); // -3, унарный минус применён к результату сложения x+2 | |
| alert( -(-3) );  // 3 |

Бинарным называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме:

|  |
| --- |
| var x = 1, y = 3; |
| alert( y - x ); // 2, бинарный минус | |

Унарный плюс, преобразует строку в число

|  |
| --- |
| var a = "2"; |
| var b = "3"; |

|  |
| --- |
|  |
| alert( a + b ); // "23", так как бинарный плюс складывает строки | |

|  |  |
| --- | --- |
| alert( +a + b ); // "23", второй операнд - всё ещё строка | |
|  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| alert( +a + +b); // 5, число, так как оба операнда предварительно преобразованы в числа  **Оператор присваения**   |  | | --- | | var a = 1; | | var b = 2; |  |  | | --- | |  | | var c = 3 - (a = b + 1); | |  |  | | --- | |  | | alert(a); // 3 | |  |  | | --- | | alert(c); // 0 |    **Инкремент** ++ увеличивает на 1:   **Декремент** -- уменьшает на 1:  i++ (называется «постфиксная форма») или ++i («префиксная форма»). |

По битовая система исчесления

По битовое и

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **a & b** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

По битовое или

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **a | b** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

По битовое исключающие или

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **a ^ b** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

По битовое не

|  |  |
| --- | --- |
| **a** | **~a** |
| 0 | 1 |
| 1 | 0 |

По битовый сдвиг

Например, 9 << 2 даст 36:

|  |
| --- |
| 9 (по осн.10) |
| = 00000000000000000000000000001001 (по осн.2) | |

|  |  |
| --- | --- |
| -------------------------------- | |
| 9 << 2 (по осн.10) |

|  |  |
| --- | --- |
| = 00000000000000000000000000100100 (по осн.2) | |
| = 36 (по осн.10)  Аналогично, -9 >> 2 даст -3, так как знак сохранен:   |  | | --- | | -9 (по осн.10) | | = 11111111111111111111111111110111 (по осн.2) | |  |  |  | | --- | --- | | -------------------------------- | | | -9 >> 2 (по осн.10) |  |  | | --- | | = 11111111111111111111111111111101 (по осн.2) = -3 (по осн.10) |   Для отрицательных чисел — результат работы разный. Например, -9 >>> 2 даст 1073741821, в отличие от -9 >> 2 (дает -3):   |  | | --- | | -9 (по осн.10) | | = 11111111111111111111111111110111 (по осн.2) | |  |  |  | | --- | --- | | -------------------------------- | | | -9 >>> 2 (по осн.10) |  |  |  |  | | --- | --- | --- | | = 00111111111111111111111111111101 (по осн.2) | | | | = 1073741821 (по осн.10) | | | var check = ACCESS\_GOODS\_VIEW | ACCESS\_GOODS\_CHANGE; // 6, 00110 | | | | | | Кодирование и по битовое сравнение | |  |  |  | | --- | --- | | var access = 30; // 11110; | | |  |  |  |  | | --- | --- | | alert(access & check); // не 0, значит есть доступ к просмотру ИЛИ изменению | | |  |  |  |  | | --- | --- | | access = parseInt("11100", 2); | | |  |  |  | | --- | | alert(access & check); // не 0, есть доступ к просмотру ИЛИ изменению | |

* parseInt("11000", 2) — переводит строку с двоичной записью числа в число.
* n.toString(2) — получает для числа n запись в 2-ной системе в виде строки.

var n=10;

var k;

k=parseInt("11000", 2);

n=n.toString(2);

console.log(k+" "+n);

**Округление**

alert( 12.345^0 ); // 12

**Логическое сравнение**

|  |
| --- |
| alert( "2" > "14" ); // true, неверно, ведь 2 не больше 14 |

Правильно было бы преобразовать их к числу явным образом. Например, поставив перед ними +:

|  |
| --- |
| alert( +"2" > +"14" ); // false, теперь правильно |
|  |

**Численное преобразование**

|  |  |
| --- | --- |
| var a = +"\n123\n"; // 123 | |
| var a = Number("\n123\n"); // 123, тот же эффект | | |
| **Значение** | **Преобразуется в…** | | |
| undefined | NaN | | |
| null | 0 | | |
| true / false | 1 / 0 | | |
| Строка | Пробелы по краям обрезаются. Далее, если остаётся пустая строка, то 0. Из непустой строки «считывается» число, при ошибке результат: NaN. | | |

alert( String(null) === "null" ); // true

var a = Number("\n123\n"); // 123, тот же эффект

Boolean(value) — в логическом контексте, можно также сделать двойным НЕ: !!value.

**Чтобы преобразовать значение к логическому типу, нужно использовать двойное отрицание: !!val или прямой вызов Boolean(val).**

**Строковое преобразование**

|  |
| --- |
| var room = { |
| number: 777, | |

|  |
| --- |
|  |
| **valueOf:** function() { return this.number; }, | |

|  |  |
| --- | --- |
| **toString:** function() { return this.number; } | |
| }; |

|  |
| --- |
|  |
| alert( +room );  // 777, вызвался valueOf | |

|  |
| --- |
|  |
| delete room.valueOf; | |

|  |
| --- |
|  |
| alert( +room );  // 777, вызвался toString | |

**У массива нет valueOf, сработает toString**

|  |
| --- |
| alert( [1,[0],2][1] ); |

Квадратные скобки после массива/объекта обозначают не другой массив, а взятие элемента.

## [Метки](http://learn.javascript.ru/break-continue" \l "%D0%BC%D0%B5%D1%82%D0%BA%D0%B8)

|  |
| --- |
| outer: |
| for (var i = 0; i < 3; i++) { | |

|  |
| --- |
|  |
| for (var j = 0; j < 3; j++) { | |

|  |
| --- |
|  |
| var input = prompt('Значение в координатах '+i+','+j, ''); | |

|  |  |
| --- | --- |
|  | |
| if (input == null) break outer; // возвращает и прерывает два цикла (\*) | | |
| } |

|  |
| --- |
| } |
| alert('Готово!'); | |

**Строки**

У строки есть свойство length, содержащее длину.

Числа

У чисел есть метод num.toFixed(n). Он округляет число num до n знаков после запятой, при необходимости добивает нулями до данной длины и возвращает в виде строки (удобно для форматированного вывода).

Если нужно число то можно применить '+' к результату n.toFixed(..).

(Math.round(price\*10))/10 ; для округления числа более корректно

|  |  |
| --- | --- |
| var price = 6.35; | |
|  |

|  |
| --- |
| alert( price.toFixed(1) ); // 6.3 |
| alert( Math.round(price\*10)/10 ); // 6.4 | |
| Math.max(a, b, c...)  Возвращает наибольший из списка аргументов  Math.min(a, b, c...)  Возвращает наименьший из списка аргументов | |

**Math.floor(m) – целое число, все что после точки отбрасывается!**

|  |
| --- |
| function getDecimal(num) { |
| return num - Math.floor(num); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| alert( getDecimal(12.5) ); // 0.5 |
| alert( getDecimal(6.25) ); // 0.25 | |

## [Создание объектов](http://learn.javascript.ru/object" \l "%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BE%D0%B2)

Пустой объект («пустой шкаф») может быть создан одним из двух синтаксисов:

|  |  |  |
| --- | --- | --- |
| 1. o = new Object(); | | |
| 2. o = {};   // пустые фигурные скобки | | | |
|  | | | |
| var person = {}; | |
|  |

|  |
| --- |
| // 1. присвоение |
| // при присвоении свойства в объекте автоматически создаётся "ящик" | |

|  |  |
| --- | --- |
| // с именем "name" и в него записывается содержимое 'Вася' | |
| person.name = 'Вася'; |

|  |
| --- |
|  |
| person.age = 25;  // запишем ещё одно свойство: с именем 'age' и значением 25 | |

|  |
| --- |
|  |
| // 2. чтение | |

|  |  |
| --- | --- |
| alert(person.name + ': ' + person.age); // вывести значения | |
|  |

|  |
| --- |
| // 3. удаление |
| delete person.name; // удалить "ящик" с именем "name" вместе со значением в нём  альтернатива   |  |  | | --- | --- | | var person = {}; | | |  |  |  |  | | --- | --- | | person['name'] = 'Вася'; // то же что и person.name | | |  |  |  |  | | --- | --- | | alert(person['name']); | | |  |  |  | | --- | | delete person['name']; | | |  | | --- | | person['день рождения!'] = '18.04.1982'; | | // запись через точку "person.день рождения! = ..." невозможна | | | | |

**Проверка существования свойства с определенным ключом.**

|  |  |
| --- | --- |
| var person = { }; | |
|  |

|  |
| --- |
| if ("age" in person) { |
| alert("Свойство age существует!"); | |

|  |
| --- |
| } |
| Или  if (obj.prop !== undefined). |
|  |

**var f={name:'Vitek',**

**surname:'Vitek111'};**

**for(key in f)**

**alert(key);**

|  |
| --- |
| var codes = { |
| "+255": "Танзания", // передаём 255 в виде строки, чтобы браузер сохранил порядок | |

|  |
| --- |
| "+7": "Россия", |
| "+38": "Украина", | |

|  |  |
| --- | --- |
| "+1": "США" | |
| }; |

|  |
| --- |
|  |
| for (var key in codes ) { | |

|  |
| --- |
| var value = codes[key]; |
| var id = +key; // ..если нам нужно именно число, преобразуем: "+255" -> 255 | |

|  |
| --- |
|  |
| alert( id + ": " + value ); // 255, 7, 38, 1 во всех браузерах | |

|  |  |
| --- | --- |
| }  Объект копируется по ссылке а не по значению | |
| var user = { name: 'Вася' }; | | |
|  |

|  |  |
| --- | --- |
| var admin = user; | |
|  |

|  |  |
| --- | --- |
| admin.name = 'Петя'; // поменяли данные через admin | |
|  |

|  |
| --- |
| alert(user.name); // 'Петя', изменения видны в user |
| Чтобы скопировать данные нужно копировать каждое значение   |  |  | | --- | --- | | function clone(obj) { | | | var obj2 = {}; |  |  | | --- | |  | | // если свойства могут быть объектами, то нужно перебирать и их | |  |  |  | | --- | --- | | for(var key in obj) obj2[key] = obj[key]; | | |  |  |  |  | | --- | --- | | return obj2; | | | } | |

**Массив**

|  |
| --- |
| // микс значений |
| var arr = [ 1, 'Имя', { name: 'Петя' }, true ]; | |

|  |
| --- |
|  |
| // получить объект из массива и тут же -- его свойство | |

|  |
| --- |
| alert( arr[2].name ); // Петя |

## [Методы pop/push, shift/unshift](http://learn.javascript.ru/array" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B-pop-push-shift-unshift)

|  |  |
| --- | --- |
| // создаем пустой массив и проверяем поддерживается ли indexOf | |
| if ( [].indexOf ) { |

|  |
| --- |
|  |
| var find = function(array, value) { | |

|  |  |
| --- | --- |
| return array.indexOf(value); | |
| } |

|  |
| --- |
|  |
| } else { | |

|  |
| --- |
| var find = function(array, value) { |
| for(var i=0; i<array.length; i++) { | |

|  |  |
| --- | --- |
| if (array[i] === value) return i; | |
| } |

|  |
| --- |
|  |
| return -1; | |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| } |
| |  |  | | --- | --- | | var arr = ["a", -1, 2, "b"]; | | |  |  |  |  | | --- | --- | | var index = find(arr, 2); | | |  |  |  | | --- | | alert(index); | |

**var arr=['1','2','3','4'];**

**alert(arr.indexOf('3')); возвращает позицию в массиве**

## [Object.keys(obj)](http://learn.javascript.ru/array-methods" \l "object-keys-obj)

В JavaScript есть встроенный метод [Object.keys(obj)](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Object/keys), который возвращает ключи объекта в виде массива. Он поддерживается везде, кроме IE<9:

В более старых браузерах аналогом будет цикл:

|  |  |
| --- | --- |
| var keys = []; | |
| for(var key in user) keys.push(key); | | |
| [Метод split](http://learn.javascript.ru/array-methods" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-split) var names = 'Маша, Петя, Марина, Василий'; | | | |
|  |

|  |  |
| --- | --- |
| var arr = names.split(', '); | |
|  |

|  |
| --- |
| for (var i=0; i<arr.length; i++) { |
| alert('Вам сообщение ' + arr[i]); | |

|  |
| --- |
| } |

У метода split есть необязательный второй аргумент — ограничение на количество элементов в массиве. Если их больше, чем указано — остаток массива будет отброшен:

|  |
| --- |
| alert( "a,b,c,d".split(',', 2) ); // a,b |

Разбивка по буквам

Вызов str.split('') разобьёт строку на буквы:

|  |  |
| --- | --- |
| var str = "тест"; | |
|  |

|  |
| --- |
| alert( str.split('') ); // т,е,с,т |

## [Метод join](http://learn.javascript.ru/array-methods" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-join)

Вызов arr.join(str) делает в точности противоположное split. Он берет массив и склеивает его в строку, используя str как разделитель.

Например:

|  |  |
| --- | --- |
| var arr = ['Маша', 'Петя', 'Марина', 'Василий']; | |
|  |

|  |  |
| --- | --- |
| var str = arr.join(';'); | |
|  |

|  |
| --- |
| alert(str); // Маша;Петя;Марина;Василий |

arr[i].charAt(0).toUpperCase() получить первый символ строки! И перевести в верхний регистр

## [Метод splice](http://learn.javascript.ru/array-methods" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-splice)

|  |  |
| --- | --- |
| var arr = ["Я", "изучаю", "JavaScript"]; | |
|  |

|  |  |
| --- | --- |
| arr.splice(1, 1);  // начиная с позиции 1, удалить 1 элемент | |
|  |

|  |  |
| --- | --- |
| alert(arr); //  осталось ["Я", "JavaScript"] | |
| Следующий пример показывает, как заменять элементы:  var arr = ["Я", "сейчас", "изучаю", "JavaScript"]; | | |
|  |

|  |  |
| --- | --- |
| // удалить 3 первых элемента и добавить другие вместо них | |
| arr.splice(0, 3, "Мы", "изучаем") |

|  |
| --- |
|  |
| alert( arr ) // теперь ["Мы", "изучаем", "JavaScript"] | |
|  | |

Метод **splice возвращает массив из удаленных элементов**:

Метод **splice также может вставлять элементы без удаления**, для этого достаточно установить deleteCount в 0:

|  |  |
| --- | --- |
| var arr = ["Я", "изучаю", "JavaScript"]; | |
|  |

|  |  |
| --- | --- |
| // с позиции 2 | |
| // удалить 0 |

|  |
| --- |
| // вставить "сложный", "язык" |
| arr.splice(2, 0, "сложный", "язык"); | |

|  |
| --- |
|  |
| alert(arr); // "Я", "изучаю", "сложный", "язык", "JavaScript" | |

Допускается использование отрицательного номера позиции, которая в этом случае отсчитывается с конца:

|  |  |
| --- | --- |
| var arr = [1, 2, 5] | |
|  |

|  |  |
| --- | --- |
| // начиная с позиции индексом -1 (предпоследний элемент) | |
| // удалить 0 элементов, |

|  |  |
| --- | --- |
| // затем вставить числа 3 и 4 | |
| arr.splice(-1, 0, 3, 4); |

|  |
| --- |
|  |
| alert(arr); // результат: 1,2,3,4,5 | |

## [Метод slice](http://learn.javascript.ru/array-methods" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-slice)

Метод slice(begin, end) копирует участок массива от begin до end, не включая end. Исходный массив при этом не меняется.

var arr2 = arr.slice(-2); с конца и до конца

Если вообще не указать аргументов — скопируется весь массив:

|  |
| --- |
| var fullCopy = arr.slice(); |

## [Сортировка, метод sort(fn)](http://learn.javascript.ru/array-methods" \l "%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-sort-fn)

**Строки сортирует строки можно сортировать и просто** .sort();

**this.SortByName=function()**

**{var tmp=function(a,b){**

**if(a.GetName()>b.GetName())**

**return 1;**

**else**

**if(a.GetName()<b.GetName())**

**return -1;**

**else**

**return 0;**

**}**

**this.obj.sort(tmp);}**

**obj.SortByName();**

**числа сортирует**

**this.SortByPrice=function()**

**{var tmp=function(a,b){return b.GetPrice()>a.GetPrice();}**

**this.obj.sort(tmp);}**

**можно так**

|  |  |
| --- | --- |
| function compareNumeric(a, b) { | |
| return a - b; |

|  |
| --- |
| } |

## [reverse](http://learn.javascript.ru/array-methods" \l "reverse)

Метод [arr.reverse()](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/reverse) меняет порядок элементов в массиве на обратный.

|  |  |
| --- | --- |
| var arr = [1,2,3]; | |
| arr.reverse(); |

|  |
| --- |
|  |
| alert(arr); // 3,2,1 | |

## [concat](http://learn.javascript.ru/array-methods" \l "concat)

Метод [arr.concat(value1, value2, … valueN)](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/concat) создаёт новый массив, в который копируются элементы из arr, а также value1, value2, ... valueN.

|  |
| --- |
| var arr = [1,2]; |
| var newArr = arr.concat(3,4); | |

|  |
| --- |
|  |
| alert(newArr); // 1,2,3,4 | |

## [indexOf/lastIndexOf](http://learn.javascript.ru/array-methods" \l "indexof-lastindexof)

## var arr = [ 1, 0, false,0];

## alert( arr.indexOf(0,2) ); // 3, вторым значением с какой позиции искать, строго по типу, возвращает номер позиции

Метод [arr.lastIndexOf(searchElement[, fromIndex])](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/lastIndexOf) ищет справа-налево: с конца массива или с номера fromIndex, если он указан.

"Blue Whale".indexOf("Whale", 5); // returns 5

# Дата и Время

new Date(year, month, date, hours, minutes, seconds, ms)

Создание даты

var now = new Date();

alert(now);

**Заметим, что год year должен быть из 4 цифр, а отсчет месяцев month начинается с нуля 0.** Например:

|  |  |
| --- | --- |
| new Date(2011, 0, 1) // 1 января 2011, 00:00:00 в местной временной зоне | |
| new Date(2011, 0) // то же самое, date по умолчанию равно 1 |

|  |
| --- |
| new Date(2011, 0, 1, 0, 0, 0, 0); // то же самое |

## [Получение компонентов даты](http://learn.javascript.ru/datetime" \l "%D0%BF%D0%BE%D0%BB%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BE%D0%B2-%D0%B4%D0%B0%D1%82%D1%8B)

Для доступа к компонентам даты-времени объекта Date используются следующие методы:

getFullYear()

Получить год(из 4 цифр)

getMonth()

Получить месяц, **от 0 до 11**.

getDate()

Получить число месяца, от 1 до 31.

getHours(), getMinutes(), getSeconds(), getMilliseconds()

getDay()

Получить номер дня в неделе. Неделя в JavaScript начинается с воскресенья, так что результат будет числом **от 0(воскресенье) до 6(суббота)**.

Существуют также UTC-варианты этих методов, возвращающие день, месяц, год и т.п. для зоны GMT+0 (UTC): getUTCFullYear(), getUTCMonth(), getUTCDay().

getTime()

Возвращает число миллисекунд, прошедших с 01.01.1970 00:00:00 UTC.

## [Установка компонентов даты](http://learn.javascript.ru/datetime" \l "%D1%83%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BE%D0%B2-%D0%B4%D0%B0%D1%82%D1%8B)

Следующие методы позволяют устанавливать компоненты даты и времени:

* setFullYear(year [, month, date])
* setMonth(month [, date])
* setDate(date)
* setHours(hour [, min, sec, ms])
* setMinutes(min [, sec, ms])
* setSeconds(sec [, ms])
* setMilliseconds(ms)
* setTime(milliseconds) (устанавливает всю дату по миллисекундам с 01.01.1970 UTC)

Все они, кроме setTime(), обладают также UTC-вариантом, например: setUTCHours().

Узнать сколько прошло дней

var now = new Date(2014,2,9);

var end = new Date(2014,2,1);

alert(((now-end)/86400000).toFixed());

toString(), toDateString(), toTimeString()

Возвращают стандартное строчное представление, не указанное в стандарте, а зависящее от браузера. Единственное требование - читаемость человеком. Метод toString возвращает дату целиком, toDateString() и toTimeString() - только дату и время соответственно.

Разница между датами до дней, часов, минут

var now = new Date(2014,2,1,21,23);

var end = new Date(2014,2,1,18,24);

var day=((now-end)/86400000).toFixed();

var hours=Math.floor(((now-end)%86400000)/3600000);

var minutes=Math.floor(((now-end)%86400000)%3600000/60000);

alert(day+" "+hours+" "+minutes);

## [Функция — это значение](http://learn.javascript.ru/function-is-value" \l "%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F-%D1%8D%D1%82%D0%BE-%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)

|  |  |
| --- | --- |
| var res = function(a,b) { return a+b }(2,2); | |
| alert(res); // 4 |

* Если функция объявлена в *основном потоке кода*, то это Function Declaration.
* Если функция создана как часть выражения, то это Function Expression.

|  |
| --- |
| // Function Expression |
| var f = function() { ... } | |

|  |
| --- |
|  |
| // Function Declaration | |

|  |
| --- |
| function f() { ... } |

Между этими двумя основными способами создания функций есть следующие различия:

|  |  |  |
| --- | --- | --- |
|  | **Function Declaration** | **Function Expression** |
| Время создания | До выполнения первой строчки кода. | Когда управление достигает строки с функцией. |
| Можно вызвать до объявления | Да (т.к. создается заранее) | Нет |
| Можно объявить в if | Нет (т.к. создается заранее) | Да |
| Можно вызывать «на месте» | Нет (ограничение синтаксиса JavaScript) | Да |

# Именованные функциональные выражения

|  |
| --- |
| var f = function factorial(n) { |
| return n ? n\*factorial(n-1) : 1; | |

|  |  |
| --- | --- |
| }; | |
|  |

|  |  |
| --- | --- |
| var g = f;  // скопировали ссылку на функцию-факториал в g | |
| f = null; |

|  |
| --- |
|  |
| alert( g(5) ); // 120, работает! | |

Изначально функция создается и присваивается, а после созданная переменная присваивает иное значение

|  |  |
| --- | --- |
| var a = 5; | |
|  |

|  |  |
| --- | --- |
| function a() { } | |
|  |

|  |
| --- |
| alert(a); Будет 5 |

if("a" in window) ошибка так как а создается во время присвоения

{a = 5;}

alert(a);

[**Замыканием**](http://en.wikipedia.org/wiki/Closure_%28computer_science%29) **функции называется сама эта функция, плюс вся цепочка LexicalEnvironment, которая при этом образуется.**

Иногда говорят «переменная берётся из замыкания». Это означает — из внешнего объекта переменных.

Можно сказать и по-другому: «замыкание — это функция и все внешние переменные, которые ей доступны».

|  |
| --- |
| var a = 1; |
| function getFunc() { | |

|  |  |
| --- | --- |
| var a = 2; | |
|  |

|  |  |
| --- | --- |
| var func = function() { alert(a); }; | |
|  |

|  |  |
| --- | --- |
| return func; | |
| } |

|  |
| --- |
|  |
| getFunc()(); вызов функции 2 выведет | |

**При создании функции с использованием new Function, её свойство [[Scope]] ссылается не на текущий LexicalEnvironment, а на window.**

|  |
| --- |
| var a = 1; |
| function getFunc() { | |

|  |  |
| --- | --- |
| var a = 2; | |
|  |

|  |  |
| --- | --- |
| var func = new Function('', 'alert(a)'); | |
|  |

|  |  |
| --- | --- |
| return func; | |
| } |

|  |
| --- |
|  |
| getFunc()();1 выведет | |

Инициализация функцией при этом создается уникальная переменная которая ни от кого не зависит

|  |  |
| --- | --- |
| function makeArmy() { | |
|  |

|  |  |
| --- | --- |
| var shooters = []; | |
|  |

|  |  |
| --- | --- |
| for(var i=0; i<10; i++) { | |
|  |

|  |  |
| --- | --- |
| var shooter = function me() { | |
| alert( me.i ); |

|  |
| --- |
| }; |
| shooter.i = i; | |

|  |
| --- |
|  |
| shooters.push(shooter); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| return shooters; | |
| } |

|  |
| --- |
|  |
| var army = makeArmy(); | |

|  |
| --- |
|  |
| army[0](); // 0 | |

|  |
| --- |
| army[1](); // 1 |

Или

|  |  |
| --- | --- |
| var shooter = (function(i) { | |
| return function() { |

|  |  |
| --- | --- |
| alert( i ); | |
| }; |

|  |
| --- |
| })(i); |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Статическая перемнная |
| |  |  | | --- | --- | | function makeCounter() { | | | return function f() { |   Отличная проверка есть ли метод!!!!!!!   |  |  | | --- | --- | | if (!f.currentCount) { | | | f.currentCount = 0; |  |  |  | | --- | --- | | } | | |  |  |  |  | | --- | --- | | return ++f.currentCount; | | | }; |  |  |  | | --- | --- | | } | | |  |  |  | | --- | | var c1 = makeCounter(); | | var c2 = makeCounter(); |  |  | | --- | |  | | alert( c1() ); // 1 | |  |  | | --- | | alert( c2() ); // 1 | | alert( c1() ); // 2 | |

# Псевдо-массив arguments

Он содержит список аргументов по номерам: arguments[0], arguments[1]…, а также свойство length.

|  |
| --- |
| function f(x) { |
| arguments[0] = 5; // меняет переменную x | |

|  |  |
| --- | --- |
| alert(x); // 5 | |
| } |

|  |
| --- |
|  |
| f(1); | |

Если вы не используете строгий режим, то чтобы переменные не менялись «неожиданно», **рекомендуется никогда не изменять arguments.**

**Методов как у массива у него нет**

**Но,** Впрочем, никто не мешает сделать обычный массив из arguments:

|  |
| --- |
| var args = []; |
| for(var i=0; i<arguments.length; i++) { | |

|  |  |
| --- | --- |
| args[i] = arguments[i]; | |
| }  Вызвать рекурсивно функцию если она не имеет названия |

setTimeout(function() {

console.log("1");

arguments.callee(); вызывает саму себя

},1000);

|  |  |
| --- | --- |
| // factorial(n) = n\*factorial(n-1) | |
| var factorial = function(n) { |

|  |  |
| --- | --- |
| return n==1 ? 1 : n\*arguments.callee(n-1); Передаются параметры} | |
| var g =factorial; |

|  |  |
| --- | --- |
| factorial = 0; // функция переместилась в переменную g | |
|  |

|  |
| --- |
| alert( g(5) ); // 120, работает! |
| Именованные аргументы Значения по умолчанию!!!   |  |  | | --- | --- | | function showWarning(width, height, title, contents, showYesNo) { | | | width = width || 200; // почти все значения - по умолчанию |  |  | | --- | | height = height || 100; |   Для удобства и лучше всего передавать в функцию объект а не значения так как его можно импользовать повторно и ненужные значения можно не передавать   |  | | --- | | var opts = { | | width: 400, | |  |  | | --- | | height: 200, | | contents: "Текст", | |  |  |  | | --- | --- | | showYesNo: true | | | }; |  |  | | --- | | function showWarning(opts) { | | var width = opts.width || 200;  // по умолчанию | |  |  |  | | --- | --- | | var height = opts.height || 100; | | |  |  |  |  | | --- | --- | | var title = optns.title || "Предупреждение"; | | |  |  |  |  | | --- | --- | | // ... | | | } Свои объекты: конструкторы и методы При объявлении в объект можно записать функцию. Она становится его методом, например:   |  | | --- | | var user = { | | name: 'Василий', | |  |  | | --- | |  | | // метод | |  |  | | --- | | sayHi: function() { | | alert('Привет!'); |  |  |  | | --- | --- | | } | | |  |  |  |  | | --- | --- | | }; | | |  |  |  |  |  |  | | --- | --- | --- | --- | | // Вызов метода ser.sayHi(); | | | | | [Доступ к объекту через this](http://learn.javascript.ru/object-methods" \l "%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF-%D0%BA-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D1%83-%D1%87%D0%B5%D1%80%D0%B5%D0%B7-this)  |  | | --- | | var user = { | | name: 'Василий', | |  |  | | --- | |  | | sayHi: function() { | |  |  |  | | --- | --- | | alert( this.name ); | | | } |  |  |  | | --- | --- | | }; | | |  |  |  | | --- | | user.sayHi(); | |  | |  | | | | | Прототип функции создается в объкте и передается в нее сам объкт  var user = { | | | name: 'Василий', | | |  |  | | --- | |  | | sayHi: function() { | |  |  |  | | --- | --- | | showName(this); // передать текущий объект в showName | | | } |  |  |  | | --- | --- | | }; | | |  |  |  |  | | --- | --- | | function showName(obj) { | | | alert( obj.name); |  |  |  | | --- | --- | | } | | |  |  |  | | --- | | user.sayHi(); |   Объекты класса  Названия функций, которые предназначены создавать объекты, как правило, начинают с большой буквы. | | Объект класса который по онклику заносит функцию и видимость ее сохраняет ко всем полям класса | | |

/\* function News()

{var div=null;

var news=null;

var curNews=0;

var create=function(parent)

{

div=document.createElement("span");

div.style.background="red";

news=["Sport","Economy","Politic","World news"];

div.innerHTML=news[0];

div.onclick=next;

parent.appendChild(div);

}

var next=function(){

if(curNews+1<news.length)

div.innerHTML=news[++curNews];

else

div.innerHTML=news[curNews=0];

}

return {create:create};

}

Функции, которые предназначены для создания объектов, называются конструкторами. Их названия пишут с большой буквы, чтобы отличать от обычных.

## [Вызов в контексте объекта](http://learn.javascript.ru/this" \l "%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2-%D0%B2-%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%B5-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%B0)

Самый распространенный случай — когда функция объявлена в объекте или присваивается ему, как в примере ниже:

|  |
| --- |
| var user = { |
| firstName: "Вася" | |

|  |  |
| --- | --- |
| }; | |
|  |

|  |
| --- |
| function func() { |
| alert( this.firstName ); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| user.sayHi = func; | |
|  |

|  |
| --- |
| user.sayHi();  // this = user |

Вызывать объект не важно как!

|  |  |
| --- | --- |
| user.a(); // Вася | |
| admin['b'](); |
|  |

### [Метод call](http://learn.javascript.ru/this" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-call)

Без метода call

|  |
| --- |
| var user = { firstName: "Вася" }; |
| var admin = { firstName: "Админ" }; | |

|  |
| --- |
|  |
| function func() { | |

|  |  |
| --- | --- |
| alert( this.firstName ); | |
| } |

|  |
| --- |
|  |
| user.a = func;  // присвоим одну функцию в свойства | |

|  |  |
| --- | --- |
| admin.b = func; // двух разных объектов user и admin | |
|  |

|  |  |
| --- | --- |
| user.a(); // Вася | |
| admin['b'](); |

С методом call

|  |
| --- |
| var user = { |
| firstName: "Василий", | |

|  |  |
| --- | --- |
| surname: "Петров" | |
| }; |

|  |
| --- |
|  |
| function getName(a, b) { | |

|  |  |
| --- | --- |
| alert( this[a] + ' ' + this[b] ); | |
| } |

|  |
| --- |
|  |
| getName.call(user, 'firstName', 'surname')  // "Василий Петров" | |

### С методом [apply](http://learn.javascript.ru/this" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-apply)

Принимает объект и массив

|  |  |
| --- | --- |
| func.call(context, arg1, arg2...) | |
| // то же что и: |

|  |
| --- |
| func.apply(context, [arg1, arg2 ... ]); |

### [«Одалживание метода»](http://learn.javascript.ru/this" \l "%D0%BE%D0%B4%D0%B0%D0%BB%D0%B6%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0)

|  |
| --- |
| function sayHi() { |
| arguments.join = [].join; // одолжили метод (1) | |

|  |
| --- |
|  |
| var argStr = arguments.join(':');  // (2) | |

|  |
| --- |
|  |
| alert(argStr);  // сработает и выведет 1:2:3 | |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| sayHi(1, 2, 3); |
| Как видно, используется this, числовые индексы и свойство length. Если эти свойства есть, то все в порядке. А больше ничего и не нужно. Подходит даже обычный объект:   |  |  | | --- | --- | | var obj = {  // обычный объект с числовыми индексами и length | | | 0: "А", |  |  | | --- | | 1: "Б", | | 2: "В", |  |  |  | | --- | --- | | length: 3 | | | }; |  |  | | --- | |  | | obj.join = [].join; | |  |  | | --- | | alert( obj.join(';') ); // "A;Б;В" | |

**Для безопасного вызова используем apply/call:**

|  |
| --- |
| function sayHi() { |
| var join = [].join; // ссылка на функцию теперь в переменной | |

|  |
| --- |
|  |
| // вызовем join с this=arguments, | |

|  |  |
| --- | --- |
| // этот вызов эквивалентен arguments.join(':') из примера выше | |
| var argStr = join.call(arguments, ':'); |

|  |
| --- |
|  |
| alert(argStr);  // сработает и выведет 1:2:3 | |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| sayHi(1, 2, 3); |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| function f(a) { | |
| alert(a); А вот нулевой элемент |

|  |  |
| --- | --- |
| var args = [].slice.call(arguments, 1); с первого элемента копируем | |
| g.apply(this, args); передаем массив |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| function g(a, b, c) { |
| alert( a + b + (c || 0) ); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| f("тест", 1, 2); |
| f("тест2", 1, 2, 3); | |

Вызываем функция для объекта

var t={name:10,surname:20};

var tmp=[];

for(var key in t)

tmp.push(key);

function f() {

g.apply(t,['name','surname']);

}

function g(a, b, c) {

alert( this[a] + this[b] + (this[c] || 0) );

}

## [Итого](http://learn.javascript.ru/this" \l "%D0%B8%D1%82%D0%BE%D0%B3%D0%BE)

Значение this устанавливается в зависимости от того, как вызвана функция:

При вызове функции как метода

|  |  |
| --- | --- |
| obj.func(...)    // this = obj | |
| obj["func"](...) |

При обычном вызове

|  |
| --- |
| func(...)        // this = window |

В new

|  |
| --- |
| new func()       // this = {} (новый объект) |

Явное указание

|  |  |
| --- | --- |
| func.apply(ctx, args) // this = ctx (новый объект) | |
| func.call(ctx, arg1, arg2, ...) |

# Преобразование объектов: toString и valueOf

**Если в объекте присутствует метод toString, который возвращает примитив, то он используется для преобразования.**

|  |  |
| --- | --- |
| var user = { | |
|  |

|  |  |
| --- | --- |
| firstName: 'Василий', | |
|  |

|  |
| --- |
| toString: function() { |
| return 'Пользователь ' + this.firstName; | |

|  |  |
| --- | --- |
| } | |
| }; |

|  |
| --- |
|  |
| alert( user );  // Пользователь Василий | |
| **Для численного преобразования объекта используется метод valueOf, а если его нет — то toString:**   |  | | --- | | var room = { | | number: 777, | |  |  | | --- | |  | | valueOf: function() { return this.number; }, | |  |  |  | | --- | --- | | toString: function() { return this.number; } | | | }; |  |  | | --- | |  | | alert( +room );  // 777, вызвался valueOf | |  |  | | --- | |  | | delete room.valueOf; | |  |  | | --- | |  | | alert( +room );  // 777, вызвался toString | | | |

# Оператор typeof, [[Class]] и утиная типизация

Как мы знаем, существует несколько *примитивных типов*:

null

Специальный тип, содержит только значение null.

undefined

Специальный тип, содержит только значение undefined.

number

Числа: 0, 3.14, а также значения NaN и Infinity

boolean

true, false.

string

Строки, такие как "Мяу" или пустая строка "".

**У всех встроенных объектов есть скрытое свойство [[Class]].**

**Таким образом проверяется что это за объект**

|  |
| --- |
| function sayHi(who) { |
| if (!arguments.length) { | |

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

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| if ( {}.toString.call(who) == '[object Array]' ) { | |
| for(var i=0; i<who.length; i++) sayHi(who[i]); |

|  |  |
| --- | --- |
| return; | |
| } |

|  |
| --- |
|  |
| alert('Привет, ' + who); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| // Использование: |
| sayHi(); // Привет | |

|  |  |
| --- | --- |
| sayHi("Вася"); // Привет, Вася | |
|  |

|  |  |  |
| --- | --- | --- |
| sayHi( ["Саша", "Петя", ["Маша", "Юля"] ] ); // Привет Саша..Петя..Маша..Юля | | |
| alert( {}.toString.call(123) ); // [object Number] |
| alert( {}.toString.call("строка") ); // [object String] | |

## [«Утиная» типизация](http://learn.javascript.ru/type-detection" \l "%D1%83%D1%82%D0%B8%D0%BD%D0%B0%D1%8F-%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F)

Проверить массив мы можем, уточнив наличие метода splice:

|  |  |
| --- | --- |
| var x = [1,2,3]; | |
|  |

|  |
| --- |
| if (x.splice) { |
| alert('Массив!'); | |

|  |
| --- |
| } |

## [Проверка типа для пользовательских объектов](http://learn.javascript.ru/type-detection" \l "%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D0%BA%D0%B0-%D1%82%D0%B8%D0%BF%D0%B0-%D0%B4%D0%BB%D1%8F-%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D1%85-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BE%D0%B2)

Синтаксис: obj instanceof Func.

|  |
| --- |
| var d = new Date(); |
| alert(d instanceof Date); // true | |

|  |
| --- |
|  |
| function f() { } | |

|  |
| --- |
| alert(f instanceof Function); // true |

Для *получения типа* есть два способа:

typeof

Хорош для примитивов и функций, врёт про null.

Свойство [[Class]]

Можно получить, используя {}.toString.call(obj). Это свойство содержит тип для встроенных объектов и примитивов, кроме null и undefined.

Для *проверки типов* есть еще два способа:

Утиная типизация

Можно проверить поддержку метода.

Оператор instanceof

Работает с любыми объектами, встроенными и созданными посетителем при помощи конструкторов: if (obj instanceof User) { ... }.

|  |
| --- |
| alert(typeof 123); // "number" |
| alert(typeof "строка"); // "string" | |

|  |  |
| --- | --- |
| alert(typeof new Date()); // "object" | |
| alert(typeof []); // "object" setTimeout и setInterval setTimeout("alert('Привет')", 1000);  **Функция-обёртка используется, чтобы кросс-браузерно передать аргументы и сохранить контекст выполнения.**   |  |  | | --- | --- | | function User(id) { | | | this.id = id; |  |  | | --- | |  | | this.sayHi = function() { | |  |  |  | | --- | --- | | alert(this.id); | | | }; |  |  |  | | --- | --- | | } | | |  |  |  |  | | --- | --- | | var user = new User(12345); | | |  |  |  |  | | --- | --- | | setTimeout(function() { | | | user.sayHi(); |  |  | | --- | | }, 1000); |   Отмена исполнения   |  |  | | --- | --- | | var timerId = setTimeout(function() { alert(1) }, 1000); | | |  |  |  | | --- | | clearTimeout(timerId); | |  | |

Для корректной задержки при повторениях функций вместо setInterval используется

|  |  |
| --- | --- |
| <input type="button" onclick="clearTimeout(timer)" value="Стоп"> | |
|  |

|  |
| --- |
| <script> |
| var i = 1; | |

|  |
| --- |
|  |
| var timer = setTimeout(function run() { | |

|  |
| --- |
| alert(i++); |
| timer = setTimeout(run, 2000); | |

|  |  |
| --- | --- |
| }, 2000); | |
|  |

|  |
| --- |
| </script> |

# Привязка функции к объекту и карринг: "bind/bindLate"

## [Привязка через замыкание](http://learn.javascript.ru/bind" \l "%D0%BF%D1%80%D0%B8%D0%B2%D1%8F%D0%B7%D0%BA%D0%B0-%D1%87%D0%B5%D1%80%D0%B5%D0%B7-%D0%B7%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5)

|  |  |
| --- | --- |
| function User() { | |
| this.id = 1; |

|  |
| --- |
|  |
| var self = this;  // сохранить this в замыкании | |

|  |
| --- |
|  |
| this.sayHi = function() { | |

|  |  |
| --- | --- |
| alert(self.id); | |
| }; |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| var user = new User(); | |
|  |

|  |
| --- |
| setTimeout(user.sayHi, 1000); // выведет "1" (ура, работает) |
|  |

## [Современный метод bind](http://learn.javascript.ru/bind" \l "%D1%81%D0%BE%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-bind)

|  |
| --- |
| function f() { |
| alert(this.name); | |

|  |  |
| --- | --- |
| } | |
|  |

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

|  |  |
| --- | --- |
| var f2 = f.bind(user); Создаем обвертку чтобы this видел | |
|  |

|  |
| --- |
| f2(); // выполнит f с this = user |

|  |  |
| --- | --- |
| Второй пример  function User() { | |
| this.id = 1; |

|  |
| --- |
|  |
| this.sayHi = function() { | |

|  |  |
| --- | --- |
| alert(this.id); | |
| }.bind(this); |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| var user = new User(); | |
|  |

|  |
| --- |
| setTimeout(user.sayHi, 1000); // выведет "1" |

# Статические и фабричные методы объектов

## [Статические свойства](http://learn.javascript.ru/static-properties-and-methods" \l "%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5-%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0)

|  |  |
| --- | --- |
| function Article() { | |
| Article.count++; |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| Article.count = 0; // статическое свойство-переменная |
| Article.DEFAULT\_FORMAT = "html"; // статическое свойство-константа | |
|  | |

## 

## [Статические методы](http://learn.javascript.ru/static-properties-and-methods#%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B)

Article в примере ниже считает количество созданных объектов, а метод Article.showCount() выводит его:

|  |  |
| --- | --- |
| function Article() { | |
| Article.count++; |

|  |
| --- |
|  |
| //... | |

|  |
| --- |
| } |
| Article.count = 0; | |

|  |
| --- |
|  |
| Article.showCount = function() { | |

|  |  |
| --- | --- |
| alert(this.count);  // (1) | |
| } |

|  |
| --- |
|  |
| // использование | |

|  |
| --- |
| new Article(); |
| new Article(); |

|  |
| --- |
| Article.showCount(); // (2) |

Здесь Article.count — статическое свойство, а Article.showCount — статический метод

## [Фабричные методы](http://learn.javascript.ru/static-properties-and-methods" \l "%D1%84%D0%B0%D0%B1%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B5-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B)

|  |
| --- |
| function User() { |
| this.sayHi = function() { alert(this.name) }; | |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| User.createAnonymous = function() { | |
| var user = new User; |

|  |  |
| --- | --- |
| user.name = 'Аноним'; | |
| return user; |

|  |  |
| --- | --- |
| } | |
|  |

|  |  |
| --- | --- |
| User.createFromData = function(userData) { | |
| var user = new User; |

|  |  |
| --- | --- |
| user.name = userData.name; | |
| user.age = userData.age; |

|  |  |
| --- | --- |
| return user; | |
| } |

|  |
| --- |
|  |
| // Использование | |

|  |
| --- |
|  |
| var guest = User.createAnonymous(); | |

|  |  |
| --- | --- |
| guest.sayHi(); // Аноним | |
|  |

|  |  |
| --- | --- |
| var knownUser = User.createFromData({name: 'Вася', age: 25}); | |
| knownUser.sayHi(); // Вася |

.

# Массив: Перебирающие методы

**[forEach](http://learn.javascript.ru/array-iteration" \l "foreach)**

Функция вызывается с параметрами: callback(item, i, arr):

|  |  |
| --- | --- |
| var arr = ["Яблоко", "Апельсин"]; | |
|  |

|  |
| --- |
| function outputItem(item, i, arr) { |
| alert(i + ": " + item + " (массив:" + arr + ")"); | |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| arr.forEach(outputItem); |
| **[filter](http://learn.javascript.ru/array-iteration" \l "filter)**  создаёт новый массив, в который войдут только те элементы arr, для которых вызов callback(item, i, arr) возвратит true.   |  |  | | --- | --- | | var arr = [1, -1, 2, -2, 3]; | | |  |  |  |  | | --- | --- | | function isPositive(number) { | | | return number > 0; |  |  |  | | --- | --- | | } | | |  |  |  |  | | --- | --- | | var positiveArr = arr.filter(isPositive); | | |  |  |  | | --- | | alert(positiveArr); // 1,2,3 | | **[map](http://learn.javascript.ru/array-iteration" \l "map)**  создаёт новый массив, который будет состоять из результатов вызова callback(item, i, arr) для каждого элемента arr.   |  |  | | --- | --- | | var arr = [1, 2, 3, 4]; | | |  |  |  | | --- | | function square(number) { | | return number \* number; |  |  |  | | --- | --- | | } | | |  |  |  |  | | --- | --- | | var squaredArr = arr.map(square); | | | alert(squaredArr); // 1, 4, 9, 16 | | | |  | | | **[every/some](http://learn.javascript.ru/array-iteration" \l "every-some)**  Метод [arr.every(callback[, thisArg])](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/every) возвращает true, если вызов callback вернёт true для каждого элемента arr.  Метод [arr.some(callback[, thisArg])](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/some) возвращает true, если вызов callback вернёт true для какого-нибудь элемента arr.   |  |  | | --- | --- | | var arr = [1, -1, 2, -2, 3]; | | |  |  |  |  | | --- | --- | | function isPositive(number) { | | | return number > 0; |  |  |  | | --- | --- | | } | | |  |  |  | | --- | | alert( arr.every(isPositive) ); // false, не все положительные | | alert( arr.some(isPositive) ); // true, есть хоть одно положительно | | | **[reduce/reduceRight](http://learn.javascript.ru/array-iteration" \l "reduce-reduceright)**   |  |  | | --- | --- | | var arr = ["a", "b", "c", "d"]; | | |  |  |  |  | | --- | --- | | function join(previousStr, currentItem, i) { | | | // прибавляем к строке очередной элемент |  |  |  | | --- | --- | | var str = previousStr + currentItem; | | | alert(str); // a, ab, abc, abcd |  |  |  | | --- | --- | | return str; | | | } |  |  | | --- | |  | | var result = arr.reduce(join, ""); // abcd | |  |  | | --- | |  | | alert("result = " + result); | | | | | | |  |   **Метод** [**arr.reduceRight**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/reduceRight) **работает аналогично, но идёт по массиву справа-налево** | |

# Запуск кода из строки: eval

**Вызов eval возвращает последнее вычисленное выражение**:

alert( eval('1+1') );

переменные могут быть изменены или дополнены:

|  |
| --- |
| var a = 0; |
| eval("a = 5"); | |

|  |
| --- |
| alert(a); |

## [Взаимодействие с внешним кодом, new Function](http://learn.javascript.ru/eval" \l "%D0%B2%D0%B7%D0%B0%D0%B8%D0%BC%D0%BE%D0%B4%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D0%B5-%D1%81-%D0%B2%D0%BD%D0%B5%D1%88%D0%BD%D0%B8%D0%BC-%D0%BA%D0%BE%D0%B4%D0%BE%D0%BC-new-function)

## Вызов new Function('a,b', '..тело..') создает функцию с указанными аргументами a,b и телом. Доступа к текущему замыканию у такой функции не будет, но можно передать локальные переменные и получить результат.

|  |  |
| --- | --- |
| var a = 2; | |
|  |

|  |  |
| --- | --- |
| // вместо обращения к локальной переменной через eval | |
| // будем принимать ее как аргумент функции |

|  |  |
| --- | --- |
| var f = new Function('x', 'alert(x)'); | |
|  |

|  |
| --- |
| f(a); // 2 |

Вычислить любое выражение нам поможет eval:

|  |  |
| --- | --- |
| var expr = prompt("Введите выражение?", '2\*3+2'); | |
|  |

|  |
| --- |
| alert(eval(expr)); |
|  |

### [Конструкция try..catch](http://learn.javascript.ru/exception" \l "%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F-try-catch)

Работает это так:

1. Выполняется код внутри блока try.
2. Если в нём возникнет ошибка, то выполнение try прерывается, и управление прыгает в начало блока catch.

Например:

|  |  |
| --- | --- |
| try { | |
|  |

|  |  |
| --- | --- |
| alert('Начало блока try');  // (1) <-- | |
|  |

|  |  |
| --- | --- |
| lalala; // ошибка, переменная не определена! | |
|  |

|  |  |
| --- | --- |
| alert('Конец блока try'); | |
|  |

|  |  |
| --- | --- |
| } catch(err) { | |
|  |

|  |  |
| --- | --- |
| alert('Управление получил блок catch!'); // (2) <-- | |
|  |

|  |
| --- |
| } |

Внутри catch получает объект с информацией, какая именно ошибка произошла. В примере выше он назван err, но можно и любое другое имя.

### [Объект ошибки](http://learn.javascript.ru/exception" \l "%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82-%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B8)

Есть несколько свойств, которые присутствуют в объекте ошибки:

name

Тип ошибки. Например, при обращении к несуществующей переменной равен "ReferenceError".

message

Текстовое сообщение о деталях ошибки.

stack

Везде, кроме IE<9, есть также свойство stack, указывающее, где в точности произошла ошибка.

### [Секция finally](http://learn.javascript.ru/exception" \l "%D1%81%D0%B5%D0%BA%D1%86%D0%B8%D1%8F-finally)

Конструкция try..catch может содержать ещё один блок: finally.

Секция finally не обязательна, но если она есть, то она выполняется всегда:

* после блока try, если ошибок не было,
* после catch, если они были.
* Блок finally срабатывает при любом выходе из try..catch, в том числе и return.
* В примере ниже, из try происходит return, но finally получает управление до того, как контроль возвращается во внешний код.

|  |  |
| --- | --- |
| function func() { | |
|  |

|  |
| --- |
| try { |
| // сразу вернуть значение | |

|  |  |
| --- | --- |
| return 1; | |
|  |

|  |
| --- |
| } catch(e) { |
| alert('Сюда управление не попадёт, ошибок нет'); | |

|  |
| --- |
| } finally { |
| alert('Вызов завершён'); | |

|  |  |
| --- | --- |
| } | |
| } |

|  |
| --- |
|  |
| alert( func() ); | |

* Это гарантирует освобождение ресурсов в любом случае
* Инициирует ошибку, которую можно поймать в catch. Объектом ошибки может быть что угодно, например число:

### [Генерация ошибки: throw](http://learn.javascript.ru/exception" \l "%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F-%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B8-throw)

|  |
| --- |
| try { |
| throw 123; | |

|  |
| --- |
| } catch(e) { |
| alert(e); // 123 | |

|  |
| --- |
| } |
| try { | |
| //... | | |

|  |  |
| --- | --- |
| throw new Error("Упс, ошибка"); | |
| //... |

|  |
| --- |
| } catch(e) { |
| alert(e.message); // Упс, ошибка | |

|  |  |
| --- | --- |
| alert(e.stack); // Где ошибка | |
| } |

# Формат JSON

* строки в двойных кавычках,
* число,
* логическое значение true/false,
* null.

|  |
| --- |
| { |
| name: "Вася",         // ошибка: ключ name без кавычек! | |

|  |  |
| --- | --- |
| "surname": 'Петров',  // ошибка: одинарные кавычки у значения! | |
| "age": 35             // .. а тут всё в порядке. |

|  |  |
| --- | --- |
| "isAdmin": false      // и тут тоже всё ок | |
| } |

## [JSON.stringify и JSON.parse](http://learn.javascript.ru/json" \l "json-stringify-%D0%B8-json-parse)

* Метод JSON.stringify(value, replacer, space) преобразует («сериализует») значение в JSON-строку.
* Метод JSON.parse(str, reviver) читает JavaScript-значение из строки.

|  |
| --- |
| var event = { |
| title: "Конференция", | |

|  |  |
| --- | --- |
| date: "сегодня" | |
| }; |

|  |
| --- |
|  |
| var str = JSON.stringify(event); | |

|  |  |
| --- | --- |
| alert( str ); // {"title":"Конференция","date":"сегодня"} | |
|  |

|  |  |
| --- | --- |
| // Обратное преобразование. | |
| event = JSON.parse(str); |
|  |

## 