***Function expression Funciton declaration***

*В случае с Function Expression - мы сами объявляем переменную с именем функции и присваиваем ей описание функции. В таком случае мы можем даже не писать имя функции. Другими словами: если объявление функции является частью какого-либо другого выражения, то такое объявление называется Function Expression.*

***Immediately-invoked function expression*** *- это тип вызова функции при котором функция вызывается сразу же после объявления.*

***this***

*this — это ссылка на объект, который «вызывает» код в данный момент.*

*Значение this чаще всего определяется тем, как вызывается функция. Когда функция вызывается как метод объекта, переменная this приобретает значение ссылки на объект, который вызывает этот метод*

***Obejct.keys(obj)***

Этот метод возвращает массив с именами перечисляемых свойств объекта:

var human = {

name: 'Иван',

eyesColor: 'black',

hairColor: 'black'

};

*console.log(Object.keys(human));*

Существует несколько способов проверить наличие свойства в объекте:

var human = {

name: 'Иван',

eyesColor: 'black',

hairColor: 'black'

};

console.log(typeof human.name != 'undefined'); //true

console.log('name' in human); //true

*console.log(human.hasOwnProperty('name')); //true*

***Теперь объекты:***

var user1 = {name: "Дима"}, // переменная содержит ссылку на объект

user2 = user1; // скопировали ссылку на объект

Здесь ситуация становится немного сложнее. При копировании переменной

* *копируется только ссылка, а объект остается в единственном экземпляре. Получается, что теперь две переменные ссылаются на один и тот же объект.*

***С массивами :***

var arr = [1, 2, 3, 4, 5, 6];

function func1(source) {

source[1] = 100;

}

func1(arr);

*console.log(arr); //выведет [ 1, 100, 3, 4, 5, 6 ]*

**Метод**

***Метод*** -это функция,которая хранится в качестве значения в свойствеобъекта и может вызываться посредством этого объекта.

var human = {

name: 'Сергей',

eyesColor: 'brown',

hairColor: 'black',

sayHello: function() {

console.log("Привет!");

}

};

human.sayHello(); //выведет Привет!

**this**

Метод должен иметь доступ к данным объекта для полноценной работы. Для доступа к объекту из метода используется ключевое слово ***this***. Оно ссылается на объект, в контексте которого вызван метод и позволяет обращаться к другим его методам и свойствам:

var human = {

name: 'Дима',

eyesColor: 'brown',

hairColor: 'black',

sayHello: function() {

console.log("Hello", this.name);

}

};

human.sayHello(); //выведет Привет! Меня зовут Сергей

**Усечение и увеличение массива**

При работе с массивами, длина массива(***свойства*** ***length***) автоматически обновляется, поэтому нам самим не приходится об этом заботиться. Но стоит упомянуть об одной детали - свойство **length** доступно не только для чтения, но и для записи. Если свойству **length** указать значение меньше текущего, то массив укорачивается до заданной длины.

Любые элементы, не попадающие в новый диапазон индексов - отбрасываются и их значения теряются, даже если потом вернуть **length** обратно - значения не будут восстановлены.

var arr = [1,2,3,4,5];

arr.length = 1; //укорачиваем массив до 1 элемента console.log(arr); //выведет [5]

arr.length = 4; //восстановим прежнее количество элементов

console.log(arr); //выведет [5, undefined, undefined, undefined]

Самым простым способом очистить массив будет: **foo.length = 0.**

**Методы**

**pop/push**

***Метод push()*** добавляет один или несколько новых элементов в конецмассива и возвращает его новую длину. ***Метод*** ***pop()*** - удаляет последний

элемент массива, уменьшает длину массива и возвращает удаленное им значение. Стоит обратить внимание на то, что оба эти метода изменяют массив, а не создают его модифицированную копию.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| var arr = []; |  |  |  |  |  |  |
| arr.push(1,2); | // foo: [1,2] | | | | Возвращает 2 | |
| arr.pop(); | // foo: [1] | | | Возвращает | | 2 |
| arr.push(3); | // | foo: | [1,3] | | Возвращает 2 | |
| arr.pop(); | // | foo: | [1] | Возвращает | | 3 |

arr.push([4,5]); // foo: [1,[4,5]] Возвращает 2

arr.pop() // foo: [1] Возвращает [4,5]

arr.pop(); // foo: [] Возвращает 1

**shift/unshift**

***Методы shift() и unshift()*** ведут себя во многом также,как**pop()**и**push()**,за исключением того, что они вставляют и удаляют элементы в начале массива. Метод **unshift()** добавляет один или несколько элементов в начало массива и возвращает новую длину массива. Метод **shift()** удаляет первый элемент массива и возвращает его значение, смещая все последующие элементы для занятия свободного места в начале массива.

|  |  |  |  |
| --- | --- | --- | --- |
| var arr = []; |  |  |  |
| arr.unshift(1); | // f:[1] | Возвращает: | 1 |
| arr.unshift(22); | // f:[22,1] Возвращает: 2 | | |
| arr.shift(); | // f:[1] | Возвращает: | 22 |

arr.unshift(3, [4, 5]); // f:[3,[4,5],1] Возвращает: 3

|  |  |  |
| --- | --- | --- |
| arr.shift(); | // f:[[4,5],1] Возвращает: 3 | |
| arr.shift(); | // | f:[1] Возвращает: [4,5] |
| arr.shift(); | // | f:[] Возвращает: 1 |

**16**

**join**

***Метод Array.join()*** используется для объединения элементов массива водну строку. Методу можно передать необязательный строковой аргумент, который будет использоваться для разделения элементов в строке. Если разделитель не задан, то при вызове метода символом-разделителем по умолчанию будет запятая.

var arr = ["Ветер", "Дождь", "Огонь"],

joined = arr.join();

console.log(joined); //выведет Ветер,Дождь,Огонь

***Метод Array.join()*** является обратным по отношению к методу**String.split()**,который создает массив путем разбиения строки нафрагменты.

var source = "Ветер,Дождь,Огонь", arr = source.split(',');

console.log(arr); //выведет [ 'Ветер', 'Дождь', 'Огонь' ]

**reverse**

***Метод Array.reverse()*** меняет порядок следования элементов в массиве напротивоположный и возвращает массив с переставленными элементами. Этот метод не создает новый массив с переупорядоченными элементами, а переупорядочивает их в уже существующем массиве.

var arr = ['Ветер', 'Дождь', 'Огонь'];

arr.reverse();

console.log(arr); //выведет [ 'Огонь', 'Дождь', 'Ветер' ]

**concat**

***Метод Array.concat()*** создает и возвращает новый массив,содержащийэлементы исходного массива, для которого был вызван метод **concat()**, последовательно дополненный значениями всех аргументов, переданных методу **concat()**. Если какой-либо из этих аргументов сам является массивом, тогда будут добавлены все его элементы. Имена массивов используются

* качестве аргументов и указываются в том порядке, в котором нужно объединить их элементы. Не изменяет исходных массив.

var arr = [1, 2, 3];

arr.concat(4, 5) //возвращает [1,2,3,4,5]

arr.concat([4, 5]); //то же самое - возвращает

[1,2,3,4,5]

arr.concat([4, 5], [6, 7]); //возвращает [1,2,3,4,5,6,7]

**sort**

***Метод Array.sort()*** сортирует элементы массива и возвращаетотсортированный массив. Если метод **sort()** вызывается без аргумента, то он сортирует элементы массива в алфавитном порядке (временно преобразует их в строки для выполнения сравнения). Метод **sort** изменяет массив.

var arr = ["Киви", "Апельсины", "Груши"];

console.log(arr.sort()); //выведет Апельсины, Груши, Киви

Как было сказано ранее - **sort** сортирует значения как строки. Из-за этого, числа становится не в порядке убывания, а в алфавитном порядке:

var arr = [10, 2, 5, 1];

arr.sort();

console.log(arr); //выведет [ 1, 10, 2, 5 ]

Обратите внимание, что 10 явно находится не на своем месте.

Для сортировки в каком-либо ином порядке, отличном от алфавитного, можно передать методу **sort()** в качестве аргумента функцию сравнения. Следует однако учесть, что функцию сравнения придется писать самим. Эта функция должна иметь два параметра, так как она устанавливает, какой из двух ее аргументов должен присутствовать раньше в отсортированном списке:

function mySort(a, b) {

if (a < b) {

return -1;

} else if (a > b) { return 1;

}

return 0; //если a == b

}

var arr = [10, 2, 5, 1];

arr.sort(mySort); //в качестве аргумента передается только имя функции

console.log(arr); //выведет [ 1, 2, 5, 10 ]

**slice**

***Метод Array.slice()*** используется для копирования указанного участкаиз массива и возвращает новый массив содержащий скопированные элементы. Исходный массив при этом не меняется.

Метод принимает два аргумента, которые определяют начало и конец возвращаемого участка массива. Метод копирует участок массива, начиная от **begin** до **end**, не включая **end**. Если указан только один аргумент, возвращаемый массив будет содержать все элементы от указанной позиции до конца массива. Можно использовать отрицательные индексы - они отсчитываются с конца массива.

var arr = [1,2,3,4,5];

arr.slice(0,3); //возвращает [1,2,3]

arr.slice(3); //возвращает [4,5] arr.slice(1,-1); //возвращает [2,3,4] arr.slice(-3,-2); //возвращает [3]

**splice**

***Метод Array.splice()*** -это универсальный метод для работы с массивами.Он изменяет массив, а не возвращает новый измененный массив, как это делают, например, методы **slice()** и **concat()**. Метод **splice** может удалять элементы из массива, вставлять новые элементы, заменять элементы. Он возвращает массив, состоящий из удаленных элементов, если ни один из элементов не был удален, вернет пустой массив. Первый аргумент указывает индекс в массиве, с которого начинается вставка или удаление элементов. Второй аргумент задает количество элементов, которые должны быть удалены из массива начиная с индекса, указанного в первом аргументе, если второй аргумент равен 0, то элементы не будут удалены. Если второй аргумент опущен, удаляются все элементы массива начиная с указанного индекса до конца массива. При использовании отрицательного номера позиции, отсчет элементов будет с конца массива.

После второго аргумента можно, через запятую, указать значения, которые будут вставлены, начиная с индекса, указанного первым аргументом. Посмотрите на примеры для лучшего понимания.

***Удаление элементов:***

var fruits = ["апельсины", "яблоки", "груши", "виноград"],

deleted = fruits.splice(2, 2);

console.log(deleted); //выведет ["груши", "виноград"] console.log(fruits); //выведет ["апельсины", "яблоки"]

***Удаление элементов и вставка новых:***

var fruits = ["апельсины", "яблоки", "груши", "виноград"],

deleted = fruits.splice(2, 2, "киви", "дыня");

console.log(deleted); //выведет ["груши", "виноград"] console.log(fruits); //выведет ["апельсины", "яблоки", "киви", "дыня"]

***Вставка новых элементов без удаления:***

var fruits = ["апельсины", "яблоки", "груши", "виноград"],

deleted = fruits.splice(2, 0, "киви", "дыня");

console.log(deleted); //выведет []

console.log(fruits); //выведет ["апельсины", "яблоки", "киви", "дыня", "груши", "виноград"]

**indexOf и lastIndexOf**

***Метод indexOf*** возвращает индекс элемента,значение которого равнозначению, переданному методу в качестве аргумента.

Если одинаковых вхождений несколько, то выбирается первый индекс. Если элемент с искомым значением не найден, то метод вернет **-1**. Внутри метода для поиска используется строгое сравнение **( === )**:

var arr = [1,2,3,3,4,5,3];

arr.indexOf(1); //вернет 0

arr.indexOf(3); //вернет 2

arr.indexOf('3'); //вернет -1

arr.indexOf(3,4); //вернет 6

arr.indexOf(35); //вернет -1

arr.indexOf(2); //вернет 1

***Метод lastIndexOf()*** тоже возвращает индекс элемента,значение которогоравно значению, переданному методу в качестве аргумента. Разница лишь в том, что метод **lastIndexOf()** выбирает индекс последнего вхождения.

var arr = [1,2,3,3,4,5,2,3];

arr.lastIndexOf(3); //вернет 7 arr.lastIndexOf(35); //вернет -1 arr.lastIndexOf(2); //вернет 6

**Методы итераторы**

**forEach**

***Синтаксис метода:***

имя\_массива.forEach(callback, thisArg)

* качестве первого аргумента указывается callback-функция, которую метод forEach() будет вызывать для каждого элемента массива. Реализацию вызываемой функции-обработчика нужно писать самим. В вызываемую функцию будет автоматически передано 3 аргумента: первый - значение текущего элемента массива, второй - индекс текущего элемента массива, и третий - сам массив. Однако, если нужно использовать только значения элементов массива, можно написать функцию только с одним параметром. Второй аргумент - thisArg (необязательный). При помощи него можно указать - на что будет указывать this внутри callback-функци

var arr = [2, 3, 4];

function foo(value) {

var sum = value \* this;

console.log(sum);

}

arr.forEach(foo, 5); //второй аргумент будет передан в качестве значения this

**filter**

***Синтаксис метода:***

имя\_массива.filter(callback, thisObject)

Метод filter() создает и возвращает новый массив, который будет содержать только те элементы массива, для которых вызов функции callback возвратит true.

function isBig(element, index, array) { //возвращает числа, которые больше или равны 10

return (element >= 10); //если значение элемента больше или равно 10 - выражение вернет true

}

var filtered = [11, 3, 7, 50, 25].filter(isBig); //filtered[11, 50, 25]

**map**

***Метод map()*** создает и возвращает новый массив,который будет состоятьиз результатов вызова функции **callback(item, idx, ar)** для каждого элемента массива.

var a = [5, 6, 7],

b = a.map(function(item, idx, arr) { return item \* item;

}); // b = [25, 36, 49]