# Объекты

## Общие понятия

Возьмём массив, допустим arr = [‘пн’,’вт’,’ср’,’чт’,’пт’].  
Чтобы обратиться к элементу массива, мы должны написать сам массив и рядом с ним ключ в квадратных скобках.  
Это может быть не удобно, т.к. первый элемент массива имеет ключ – **0** и т.д.  
Удобнее было бы начать отсчёт допустим с **1.** В этом нам поможет объект (или в других ЯП ассоциативный массив, хеш).  
В объекте мы сами задаём ключи и значения которые нам нужны. Объекты указываются в фигурных скобках { } и **имеют формат ключ:значение.**

## Синтаксис

let obj = {1: ‘пн’, 2: ‘вт’, 3: ‘ср’}  
console.log(obj[1]); //выведет ‘пн’;

## Вывод всего объекта

Чтобы вывести весь объект и увидеть реальные его значения, нужно использовать функцию console.log

# let obj = {1: 'a', 2: 'b', 3: 'c'}; alert(obj); // выведет [Object object] console.log(obj); // выведет сам объект

## Строковые ключи

В объектах ключи не обязательно должны быть в числовом формате. Они могут быть и строковыми, тогда при выводе, ключ нужно будет писать в **строковом виде.**

let obj = {key1: 'a', key2: 'b', key3: 'c'};  
console.log(obj[‘key1’]); // выведет ‘a’;

## Заполнение значениями

Объекты заполняются точно так же, как и массивы

let obj = {};  
obj[‘key1’] = ‘a’;  
obj[‘key2’] = ‘b’;  
obj[‘key3’] = ‘c’;  
console.log(obj); // Выведет {key1: ‘a’, key2: ‘b’, key3: ‘c’}  
  
**И альтернативный синтаксис**let obj = {};  
obj.key1 = ‘a’;  
obj.key2 = ‘b’;  
obj.key3 = ‘c’;  
console.log(obj); // Выведет {key1: ‘a’, key2: ‘b’, key3: ‘c’}

## Ограничения на строковые ключи

Строковые ключи могут быть любыми, **кроме тех, которые начинаются с цифры, имеют дефис, пробел и т.д.**

Однако, такие ключи возможны, **но их нужно писать в кавычках.** Это не приветствуется, т.к. лучше всего обозначать кавычки без ключей, где это возможно!

let obj = {‘1key’: ‘a’, ’key-2’: ‘b’, key3: ‘c’);  
console.log(obj[‘1key’]); // выведет ‘a’  
console.log(obj[‘key-2’]); // выведет ‘b’  
console.log(obj[‘key3’]); // выведет ‘c’

## Альтернативный синтаксис

Объекты так же имеют еще один метод вывода по мимо такого **объект[ключ].** Так же ещё можно обратиться к элементу вот так **объект.ключ.**Данный метод обращения имеет ограничение. **НЕЛЬЗЯ ОБРАЩАТЬСЯ, ЕСЛИ ЕСТЬ КЛЮЧИ, КОТОРЫЕ ПИШУТСЯ В КАВЫЧКАХ (объект.’ключ’)**Этот метод называется *обращение через свойство объекта.*

let obj = {key1: 1, key2: 2, key3: 3};  
console.log(obj.key1); // Вернёт – 1;  
console.log(obj.key2); // Вернёт – 2;  
console.log(obj.key3); // Вернёт – 3;

# Методы работы с массивами

## Метод reverse

Метод **reverse** изменяет порядок элементов в массиве на обратный.

Метод **изменяет исходный массив** (меняет местами в обратном порядке элементы) и возвращает новый массив.

массив.**reverse();**  
let arr = [‘a’,’b’,’c’];  
arr.reverse;  
console.log(arr); // Вернёт [‘c’,’b’,’a’];

## Метод push

Метод **push** добавляет элементы массива в конец указанного массива. Число не ограничено.

Метод **изменяет исходный массив** и возвращает новый массив.

массив.**push(**элемент, элемент, элемент …**);**  
let arr = [‘a’,’b’,’c’];  
arr.push(‘j’,’g’,’k’);  
console.log(arr); // Вернёт [‘a’,’b’,’c’,’j’,’g’,’k’]

## Метод unshift

Метод **unshift** добавляет элементы массива в начало указанного массива. Число не ограничено.

Метод **изменяет исходный массив** и возвращает новый массив.

массив.**unshift(**элемент, элемент, элемент …**);**  
let arr = [‘a’,’b’,’c’];  
arr.push(‘j’,’g’,’k’);  
console.log(arr); // Вернёт [‘j’,’g’,’k’,’a’,’b’,’c’]

## Метод shift

Метод **shift** удаляет первый элемент массива.

Метод **изменяет исходный массив** и возвращает удаленный элемент.

массив.**shift();**  
let arr = [‘a’,’b’,’c’];  
arr.shift();  
console.log(arr); // Вернёт [‘b’,’c’]  
  
let arr = [‘привет’,’мой’,’мир’];  
console.log(arr.shift()); //Вернёт ‘привет’

## Метод pop

Метод **pop** удаляет последний элемент массива.

Метод **изменяет исходный массив** и возвращает удаленный элемент.

массив.**pop();**  
let arr = [‘a’,’b’,’c’];  
arr.pop();  
  
console.log(arr); // Вернёт [‘a’,’b’]  
let arr = [‘привет’,’мой’,’мир’];  
console.log(arr.pop()); //Вернёт ‘мир’

## Метод slice

Метод **slice** возвращает указанную часть массива.

**Первый параметр -** номер элемента с которого начать, а **Второй параметр -** номер элемента на котором закончить (**! не включая его !)**.  
**Второй параметр необязательный,** если его не указать, то вырезаться будет до конца массива.  
**Второй параметр может быть отрицательным,** в таком случае будет браться элемент с конца массива. (-1 - последний, -2 - предпоследний и т.д.)

**Метод не изменяет исходный массив.**

массив.**slice(**откуда отрезать, докуда отрезать**);**  
let arr = [‘a’,’b’,’c’,’d’,’e’,’f’];  
let newArr = arr.slice(1,4);  
console.log(arr.slice(0,4)); // Вернёт [‘a’,’b’,’c’,’d’]  
console.log(newArr); // Вернёт [‘b’,’c’,’d’]  
console.log(arr); //Вернёт [‘a’,’b’,’c’,’d’,’e’,’f’]

## Метод splice

Метод **splice** удаляет или добавляет элементы в массив. Можно только **удалять** или только **добавлять**. Также можно **и то, и другое** одновременно.

**Первый параметр -** номер элемента с которого начать, а **Второй параметр -** номер элемента на котором закончить.

**Второй параметр необязательный,** если его не указать, то элементы вырезаться не будут, только добавляться.  
  
**Далее идут через запятые элементы,** которые нужно добавить в массив, вместо удалённых. Если удаления не было (второй параметр == 0), то элементы вставляются с той позиции, которая указана первым параметром.   
  
**Первый параметр может быть отрицательным,** в таком случае будет браться элемент с конца массива. (-1 - последний, -2 - предпоследний и т.д.)

**Метод изменяет сам массив и возвращает массив удаленных элементов.**

массив.**splice(**откуда удалять, сколько элементов удалять, элемент для вставки, элемент для вставки, ...**);**  
let arr = [‘a’,’b’,’c’,’d’,’e’,’f’];  
let deleteElemArr = arr.splice(1,3);  
console.log(deleteElemArr); // Вернёт [‘b’, ‘c’, ‘d’]  
console.log(arr); // Вернёт [‘a’,’e’,’f’]  
  
let arr = [‘a’,’b’,’c’,’d’,’e’,’f’];  
let deleteElemArr = arr.splice(1,3,’Да’,’Нет’,’Не знаю’);  
console.log(deleteElemArr); // Вернёт [‘b’, ‘c’, ‘d’]  
console.log(arr); // Вернёт [‘a’,’Да’,’Нет’,’Не знаю’,’e’,’f’]  
  
let arr = [‘a’,’b’,’c’,’d’,’e’,’f’];  
let deleteElemArr = arr.splice(1,0,’g’,’d’);  
console.log(deleteElemArr); // Вернёт [‘a’,’b’,’c’,’d’,’e’,’f’]  
console.log(arr); // Вернёт [‘a’,’g’,’d’,’b’,’c’,’d’,’e’,’f’]

## Метод sort

Метод **sort** производит сортировку массива в лексикографическом порядке.

Также можно указать параметром собственную функцию для сортировки, но не **обязательно.**

**Метод возвращает отсортированный массив.**

массив.**sort(**function()**);**  
let arr = [‘g’,’x’,’h’,’e’,’a’,’p’];  
let sortArr = arr.sort();  
console.log(sortArr); // Вернёт [‘a’,’e’,’g’,’h’,’p’,’x’]

## Функция Object.keys

Функция **Object.keys** преобразует ключи объекта в массив.

**Функция возвращает массив.**

Object.keys**(**объект**);**  
let obj = {a: 1, b: 2, c: 3};  
let arrKeys = Object.keys(obj);  
console.log(arrKeys); // вернёт [‘a’,’b’,’c’]

# Пользовательские функции

## Синтаксис

**Функция** создается при помощи конструкции:

function название функции(параметры через запятую) {  
 код, выполняемый функцией;   
}

Теперь по порядку:

1. **function -** служебное слово для функции в языке JavaScript.
2. **Название функции -** функция должна иметь свое название, чтобы её потом вызывать по нему!
3. **(параметры через запятую) -** параметры это обычные переменные. Их можно **не указывать**, можно указать **один** или **несколько через запятую**
4. **{} -** операторские скобки для функции. Внутри них расположен код конкретно для этой функции!

## Вызов функции

**Функция** вызывается по имени. Круглые скобки обязательны после вызова, даже, если нет параметров.

function hello() {  
 alert(‘Привет Мир!’);  
}  
hello();  
hello();  
**Выведет диалоговое окно с надписью Привет Мир! два раза.**

Как видно из примера **функцию** можно вызывать неоднократно в разных местах.

## Пример

До этого **функцию вызывали** без параметров. Ниже видно как она работает с параметрами.

function hello(text) {  
 alert(text);  
}  
hello(‘Привет Мир!’);  
hello(‘Как дела?’);  
Выведет диалоговое окно с надписью Привет Мир!, а затем еще одно с надписью Как дела?

Из примера мы видим, что параметр присваивается в скобках, а значение мы ему даём непосредственно при вызове функции.

## Инструкция return

Чаще всего функция не должна выводить окна через alert(), а возвращать какие-то значения.

function hello(text) {  
 return text;  
}  
alert(hello(‘Привет Мир!’));  
**Выведет диалоговое окно с надписью Привет Мир!**

Как видно из примера, функция возвращает значение, которое мы ей даем при вызове и выводит диалоговое окно alert().

**Далее более сложная функция, которая будет возвращать строку: Привет %имя человека%.**

function hello(name) {  
 let text = ‘Привет ’+name+’!’;  
 return text;  
}  
  
let message = hello(‘Дима’);  
alert(message);  
alert(hello(‘Дима’)+’ ’+hello(‘Коля’)+’ ‘+hello(‘Петя’));  
  
**Выведет два диалоговых окна.   
В первом: Привет Дима!  
Во втором: Привет Дима Привет Коля Привет Петя**

# Многомерные массивы

## Синтаксис и вывод элементов

Элементы массивов могут быть не только строки или числа, но и так же другие **массивы.** Тогда получится массив массивов или *многомерный массив*.

Рассмотрим **синтаксис написания.**

let arr = [ [‘a’,’b’,’c’], [‘d’,’e’,’f’], [‘g’,’h’,’i’] ];

Таким образом у нас получились массивы в одном массиве. Напишем более понятно глазу

let arr = [  
[‘a’,’b’,’c’],  
[‘d’,’e’,’f’],  
[‘g’,’h’,’i’],  
];

Массивы не обязательно могут быть двухмерными как из примера выше, они могут быть трёхмерными (массив массивов массивов), четырёхмерными и т.д. **Нет ограничений.**

Теперь чтобы вывести элемент из массива, нужно будет обращаться с помощью двух (трёх, четырёх), в зависимости от массива, скобок. **массив[ключ][ключ].**

let arr = [  
[‘a’,’b’,’c’],  
[‘d’,’e’,’f’],  
[‘g’,’h’,’i’],  
];

console.log(arr[0][1]); // Выведет ‘b’;  
console.log(arr[1][0]); // Выведет ‘d’;

## Произвольные массивы

Массивы могут иметь разный **вид**. Необязательно такой, который показан выше.

Например:  
let arr = [

[

'a', 'b', [1, 2, 3], [4, 5],

],

[

'd', ['e', 'f'],

],

];

Как видно из массива, он не совсем *“правильный”*. В нём обычные строки располагаются рядом с вложенным массивом.

## Вывод элементов массива с использованием циклов

Чтобы вывести элементы многомерного массива, нужно **использовать цикл в цикле (цикл в цикле в цикле – трёхмерные массивы).**

В циклах начинается последовательный перебор с увеличением значений счётчика по мере достижения длины массива.

C помощью конструкции for (of)  
let arr = [[1, 2, 3, 4, 5], [6, 7, 8], [9, 10]];  
for (let subArr of arr) {

for (let elem of subArr) {

console.log(elem);

}

}; // Выведет последовательно элементы в консоли.

C помощью полноценной конструкции  
let arr = [[1, 2, 3, 4, 5], [6, 7, 8], [9, 10]];  
for (let i = 0; i < arr.length; i++) {

for (let j = 0; j < arr[i].length; j++) {

console.log(arr[i][j]);

}

}; // Выведет последовательно элементы в консоли.

## Заполнение массива

Чтобы заполнить многомерный массив, нужно присваивать элементам значения, которые нужны в цикле.

В первом цикле, нужно создавать пустой массив, а во втором присваивать создавать элементы и присваивать им значения.

let arr = [];

for (let i = 0; i < 3; i++) {

arr[i] = []; // создаем подмассив

for (let j = 0; j < 3; j++) {

arr[i][j] = j + 1; // заполняем подмассив числами

}

}

console.log(arr);

## Потенциальные проблемы

Рассмотрим код выше.  
В нём есть строчка **arr[i] = [];**Здесь важно объявлять подмассив, потому что в следующем цикле при обращении к элементу массива, консоль выдаст ошибку. Это случается, потому что не создан подмассив, а мы обращаемся к элементу, как к элементу массива и поэтому массив вернёт **undefined,** потому что, мы не указали, что он является массивом.

## Заполнение массива числами по порядку

В примерах выше, числа были одинаковыми и ограничивались числом элементов, заданным как переменой **j.**

Теперь сделаем элементу по порядку. Мы подключим счётчик, который не будет сбрасываться, а только увеличиваться от числа циклов.  
Для этого есть 3 способа написания цикла.

1ый способ самый длинный, но самый понятный глазу

let arr = [];

let k = 1; // счетчик

for (let i = 0; i < 3; i++) {

arr[i] = [];

for (let j = 0; j < 3; j++) {

arr[i][j] = k; // записываем счетчик

k++; // увеличиваем счетчик

}

}

console.log(arr);

2ой способ краткий. Тут мы объявляем счётчик внутри цикла, как i и увеличиваем его во втором цикле, как j.

let arr = [];

for (let i = 0, k = 1; i < 3; i++) {

arr[i] = [];

for (let j = 0; j < 3; j++, k++) {

arr[i][j] = k;

}

}

console.log(arr);

3ий способ имеет место быть, но лучше его не использовать, т.к. можно запутаться с увеличением счётчика k.

let arr = [];

for (let i = 0, k = 1; i < 3; i++) {

arr[i] = [];

for (let j = 0; j < 3; j++) {

arr[i][j] = k++;

}

}

console.log(arr);  
Тут важно понимать, что счётчик нужно увеличивать после выполнения (k++), а не сразу как вызываем его (++k). Если сделать вторым способом, то отсчет начнется с 2 и т.д., что не совсем верно.