# 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;

# Математические методы

## Math.pow

Данный метод возводит число в степень, причем степень явно указывается вторым параметром. **Степень может быть отрицательной (по правилам математики).**

Math.pow(число, степень);

## Math.sqrt

Данный метод находит квадратный корень числа. Параметром принимает только положительные числа, иначе вернёт **NaN**.

Math.sqrt(положительное число);

## Math.round

Данный метод округляет число до ближайшего целого числа по правилам математики.  
Параметром принимает число.

Math.round(число);

## Math.ceil

Данный метод округляет число до целого всегда в большую сторону.

Math.ceil(число);

## Math.floor

Данный метод округляет число до целого всегда в меньшую сторону.

Math.floor(число);

## toFixed

Данный метод округляет число и параметром указывается число знаков после запятой.  
**Если не передавать параметр, то округлится до целого числа по правилам математики.**

Число.toFixed(количество знаков после запятой);

## toPrecision

Данный метод преобразует число в число с указанной длинной в параметре.  
**Отличием данного метода является то, что он округляет не только дробную часть, но и целую.  
Округление идёт по правилам математики, а если это невозможно, тогда переводит в экпоненциальную форму.**

Экспоненциальная форма - это числа вида **1.2e+3**. Чтобы преобразовать в нормальную форму нужно сделать так**: 1.2e+3 = 1.2 \* 10\*3 = 1.2 \* 1000 = 1200.**

Число.toPrecision(длина);

## Math.max

Данный метод находит максимальное число из переданных ему чисел в параметре **через запятую.**Если в функцию ничего не передано, **то он вернёт –Infinity(минус бесконечность)**

По умолчанию функция не работает с массивами, но есть хитрость, которая позволяет это исправить: **Math.max.apply(null, arr)**, где arr – произвольный массив.

Math.max(число1, число2, число3 …);  
Math.max.apply(null, arr); //Для работы с массивами.

## Math.min

Данный метод находит минимальное число из переданных ему чисел в параметре **через запятую.**Если в функцию ничего не передано, **то он вернёт Infinity(плюс бесконечность)**

По умолчанию функция не работает с массивами, но есть хитрость, которая позволяет это исправить: **Math.min.apply(null, arr)**, где arr – произвольный массив.

Math.min(число1, число2, число3 …);  
Math.min.apply(null, arr); //Для работы с массивами.

## Math.random

Данный метод выводит рандомное **дробное** число от 0 до 1.

Math.random();

Но что делать, если нужно получить не дробное а целое?  
**ЛАЙФХАК:**

Чтобы получить целое число, нужно воспользоваться специальной функцией, которой мы передадим промежуток от минимум до максимум.

function getRandomArbitary(min,max) {

return Math.random()\*(max-min)+min;

}// функция вывода рандомного числа в промежутке

function getRandomInt(max,min) {

return Math.floor(Math.random()\*(max-min+1))+min;

}// функция выводи целого рандомного числа в промежутке

## Math.abs

Данный метод выводит модуль числа, переданного в параметре. Делать из отрицательного числа – положительное.

Math.abs(число);

# Строковые методы

## Метод length

Свойство **length** позволяет узнать длину строки. Под длиной понимается количество символов в ней.

строка.length;

let str = 'dasdzxc';

alert(str.length); //вернёт 7

## Методы toUpperCase, toLowerCase

**Метод toUpperCase** производит преобразование строки в верхний регистр. **При этом возвращается новая строка, а исходная строка не меняется.**

строка.toUpperCase();

let str = 'Hello World';

alert(str.toUpperCase); // вернёт ‘HELLO WORLD’

**Метод toLowerCase** производит преобразование строки в нижний регистр. **При этом возвращается новая строка, а исходная строка не меняется.**

строка.toLowerCase();

let str = 'Hello World';

alert(str.toLowerCase); // вернёт ‘hello world’

## Метод substr

Метод **substr** возвращает подстроку из строки (**исходная строка при этом не меняется).**

Первый параметр задает номер символа, с которого нужно начать отрезать (нумерация начинается с нуля), а второй сколько символов нужно отрезать.

Первый параметр может принимать отрицательные значения, в таком случае отсчет будет идти с конца строки. (**-1 последний символ, -2 предпоследний и т.д.)**

Второй параметр не обязательный, если его не указывать, то будет вырезаться до конца строки.

строка.substr(откуда начинать резать, [сколько символов отрезать]);

## Метод substring

Метод **substring** возвращает подстроку из строки (**исходная строка при этом не меняется).**

Первый параметр задает номер символа, с которого нужно начать отрезать (нумерация начинается с нуля), а второй номер символа, на котором закончить вырезание (**при этом этот символ не учитывается в вырезании)**

Второй параметр не обязательный, если его не указывать, то будет вырезаться до конца строки.

**Если первый параметр больше второго, то метод ведёт себя так как будто они поменялись местами.**

Если какой-то из параметров больше длины строки, он считается равным длине строки.

Отрицательные значения приравниваются к нулю.

строка.substring(откуда начинать резать, [до какого символа резать]);

## Метод slice

Такой же как **substring,** только вторым параметром можно указывать отрицательные значения и это будет как в методе **substr.**

строка.slice(откуда отрезать, [докуда отрезать]);

/\* Метод indexOf осуществляет поиск подстроки в строке.

Метод вернет позицию первого совпаднеия, а если оно не найдено, то -1.

Вторым параметром можно(но не обязательно) передать номер символа, откуда следует начинать поиск.

Метод чувствителен к регистрку символов.

Синтаксис:

строка.indexOf(что ищем, откуда начать поиск);

let str = 'Hello World';

alert(str.indexOf('Hello'));

Вернёт 0 \*/

/\*===================================\*/

/\* Метод split осуществляет разбиение строки в массив по указанному разделителю.

Разделитель указывается первым необязательным параметром. Если он не задан - вернется вся строка. Если он зада в пустые кавычки - '', то каждый символ строки попадает в отдельный элемент массива.

Вторым необязательным параметром можно указать максимальное количество элементов в получившемся массиве.

Синтаксис:

строка.split(разделитель, максимальное количество элементов);

let str = 'Мне-очень-нравится-JavaScript',

arr = str.split('-');

document.write(arr);

Вернет Мне,очень,нравится,JavaScript\*/

/\*===================================\*/

/\*Метод replace осуществляет поиск и замену частей строки.

Первым параметром принимается подстрока, которую заменяем, а вторым - подстрока, на которую заменяем.

Синтаксис:

строка.replace(что заменяем, на что заменяем);

let str = 'Hello World';

alert(str.replace('Hello', 'Good Bye'));

Вернёт Good Bye World \*/

/\*===================================\*/

/\* Функция join массив в строку с указанным разделителем.

Синтаксис:

строка.join(разделитель);

# 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.

# Многомерные объекты

## Синтаксис

Многомерными могут быть не только массивы, но и объекты. **Например:**

let obj = {

a: {

key1: 'a1',

key2: 'a2',

key3: 'a3',

},

b: {

key1: 'b1',

key2: 'b2',

key3: 'b3',

},

c: {

1: 'c1',

2: 'c2',

3: 'c3',

},

}

Т.к. **объекты** - это те же **массивы**, но только разница в том, что объект – ассоциативный массив и тут имеются ключи.

Синтаксис точно такой же, как и в массивах.  
console.log(obj[‘a’][‘key1’]) //выведет ‘a1’  
  
Так же можно выводить, обращаясь к свойствам  
console.log(obj.a.key1) //выведет ‘a1’  
  
Можно комбинировать  
console.log(obj.a[‘key1]) //выведет ‘a1’  
console.log(obj[‘a’].key1) //выведет ‘a1’  
  
Но, если ключ является обычным числом, то обращаться к нему, как к свойству – нельзя  
console.log(obj.c.1) //выведет ошибку  
console.log(obj.c[1]) //выведет ‘c1’

## Перебор элементов многомерного объекта

Дан объект:

let obj = {

a: {

1: 'a1',

2: 'a2',

3: 'a3',

},

b: {

1: 'b1',

2: 'b2',

3: 'b3',

},

c: {

1: 'c1',

2: 'c2',

3: 'c3',

},

}

Чтобы вывести из него элементы, нужно использовать цикл (конструкцию for in)

for (let key in obj) {

let subObj = obj[key];

for (let subKey in subObj) {

console.log(subObj[subKey]);

}

}

## Ключи из переменных для объектов

Дан объект:

let obj = {

'ru': ['пн', 'вт', 'ср', 'чт', 'пт', 'сб', 'вс'],

'en': ['mn', 'ts', 'wd', 'th', 'fr', 'st', 'sn'],

};

Выведем из него какой-нибудь день недели

console.log(obj[‘ru’][2]) //вернёт ‘ср’

Давайте так же попробуем, но предварительно записав всё в переменные

let lang = ‘ru’;  
let day = 2;  
console.log(obj[lang][day]); //вернёт все тот же ‘чт’

**Так же тут есть нюансы**

1. Неправильно обращаясь к массиву с переменными саму переменную брать в кавычки

console.log(obj[‘lang’][‘day’]; //так делать неправильно

1. Неправильно так же обращаться к массиву с переменными и использовать переменную как свойство объекта

console.log(obj.lang[day]); // так делать не правильно

# Комбинация массив+объект и наоборот

## Синтаксис

Можно комбинировать массивы и объекты, а затем обращаться к ним как обычно.

Вот пример комбинации массив в объекте  
let days = {

'ru': ['пн', 'вт', 'ср', 'чт', 'пт', 'сб', 'вс'],

'en': ['mn', 'ts', 'wd', 'th', 'fr', 'st', 'sn'],

};  
console.log(days[‘ru’][0]); //выведет ‘пн’  
console.log(days[‘en’][2]); //выведет ‘wd’  
  
Вот пример комбинации объекта в массиве  
let users = [

{

name: 'name1',

age: 31,

},

{

name: 'name2',

age: 32,

},

{

name: 'name3',

age: 33,

},

];  
console.log(users[2]); // выведет {name: ‘name3’, age: 32}  
console.log(users[1].name); // тут мы хотим узнать имя и нам выдаст ‘name2’

## Перебор значений в цикле

Дан массив с пользователями:

let users = [

{

name: 'name1',

age: 31,

},

{

name: 'name2',

age: 32,

},

{

name: 'name3',

age: 33,

},

];  
  
Чтобы перебрать данный массив можно использовать классический способ перебора массива или конструкцию for of  
  
for (user of users) {  
document.write(user.name+' '+user.age+'<br>');  
}; // выведет на экран браузера данные об пользователях

Пускай данная сложная структура, состоящая из массивов и объектов

let students = {

'group1': ['student11', 'student12', 'student13'],

'group2': ['student21', 'student22', 'student23'],

'group3': ['student31', 'student32'],

};  
  
Чтобы перебрать значения элементов этой структуры, нужно поочерёдно использовать способы перебора. Например первая конструкция у нас – это объект, т.е. используем for in цикл, далее идет массив – используем for of цикл  
  
for (let group in students) {  
 for (let name of students[group]) {  
 console.log(name); //выведет имена студентов  
 }  
}

# Добавление элементов в многомерные массивы

Дан массив с пользователями

let users = [

{

name: 'name1',

age: 31,

},

{

name: 'name2',

age: 32,

},

{

name: 'name3',

age: 33,

},

];

Чтобы добавить в него элементы (например еще один объект с данными), нужно использовать метод **push.**

users.push ({

name: ‘name4’,  
 age: 34

});

Таким образом у нашего массива появится новый объект с данными, переданными выше.

# Добавление элементов в многомерные объекты

Дан объект со студентами:

let students = {

'group1': ['student11', 'student12', 'student13'],

'group2': ['student21', 'student22', 'student23'],

'group3': ['student31', 'student32'],

};  
Чтобы добавить сюда ещё одного студента, нужно обратиться к массиву по его ключу в объекте  
students.group1.push(‘student14’);  
  
Далее, можно создать ещё одну группу студентов и поместить туда нового студента  
  
students.group4 = []; //Обязательно объявляем массив, иначе потом при пуше выдаст ошибку  
students.group4.push(‘student41’);

**Попробуем усложнить хранение**

Пусть объект хранит группы, а группы подгруппы, в которых будут уже студенты

let students = {

'group1' {

'subgroup11': [

'student111', 'student112', 'student113'],

'subgroup12': [

'student121', 'student122', 'student123'],

},

'group2' {

'subgroup21': [

'student211', 'student212', 'student213'],

'subgroup22': [

'student221', 'student222', 'student223'],

},

'group3' {

'subgroup31': [

'student311', 'student312', 'student313'],

'subgroup32': [

'student321', 'student322', 'student323'],

},

};

Далее решения над этим массивом будут в задачнике!

## Автоматическое добавление

Дан объект со студнетами:

let students = {

'group1': ['student11',

'student12', 'student13'],

'group2': ['student21',

'student22', 'student23'],

'group3': ['student31', 'student32'],

};

Чтобы каждый раз не парится при добавлении группы над тем, не забыли ли мы объявить её как массив, можно написать проверку, которая будет проверять сначала является ли группа массивом, если нет то создавать массив и добавлять данные, а если да, то просто добавлять данные.

if (students[‘group4’] !== undefined) {

students[‘group4’].push(‘student41’);  
}  
  
else {  
 students[‘group4’] = [];  
 students[‘group4’].push(‘student41’);  
}  
  
Немного сократим  
  
if (students[‘group4’] === undefined) {  
 students[‘group4’] = [];  
}  
students[‘group4’].push(‘student41’);

Как работает этот сокращённый код? Он проверяет, если **массив не объявлен,** то она сначала объявит его, а потом добавит строку. Тут не важно, какой исход, в любом случае будет добавлена строка, потому что она вне ифа, но иф стоит перед методом добавления, соответственно он выполнится первым.

Теперь оформим этот в функции, для передачи параметров сразу из функции.

function addStudent (group, name) {

if (students[group] === undefined) {  
 students[group] = [];  
}

students[group].push(name);  
};  
  
addStudent(‘group5’,’student51’);  
Таким образом мы добавим новую группу 5 и студента 51.

# Удобное хранение данных

## Уникальные значения

Дан произвольный массив или нужно составить объекты в массиве. Допустим в нем есть ключи: имя, фамилия, id.  
Отлично, примерно так он будет выглядеть:

let users = [

{

id: 11,

name: 'name11',

surname: 'surname11',

},

{

id: 14,

name: 'name14',

surname: 'surname14',

},

{

id: 17,

name: 'name17',

surname: 'surname17',

},

];

При такой конструкции очень удобно обратиться к ключам, всего лишь перебрав массив и указав на ключи явно, мы получим нужные данные. **Например:**

for (user of users) {

document.write(user.id + ' ' + user.name + ' ' + user.surname + '<br>');

}  
Таким образом на экране мы получим данные о всех пользователях.

**Да, это очень удобно!**

Но, что если придется вывести пользователя с определенным **id.** Допустим это будет id = 14. Конструкция будет такова:

for (user of users) {

if(user.id === 14) {

console.log(user);  
 break;

}

}  
В итоге он вернёт нам объект значений имени и фамилии у id = 14.

Уже конструкция выглядит **не такой удобной, так как приходится делать проверки**. Давайте преобразуем массив в объект, а за ключи возьмём **id**, как мы узнали ранее, ключи должны быть уникальны, поэтому **id** то, что нужно!

let users = {

11: {

name: 'name11',

surname: 'surname11'

},

14: {

name: 'name14',

surname: 'surname14'

},

17: {

name: 'name17',

surname: 'surname17'

}

};

Теперь чтобы вывести тот же объект, достаточно напрямую обратиться по ключу (это в 3 раза меньше кода и меньше вероятность путаницы)

console.log(users[14]);  
Данная строчка вернёт тот же самый объект!

Теперь оформим вывод всех пользователей

for (id in users) {

let user = users[id];

document.write(id + ' ' + user.name + ' ' + user.surname+'<br>');

};  
Да, теперь код вывода всех стал чуть больше, но это по прежнему удобно и структурировано!

## Даты

Допустим, у нас есть данный объект с датами и делами:

let affairs = {

'2018-11-29': ['массив дел'],

'2019-12-28': ['массив дел'],

'2018-11-30': ['массив дел'],

'2019-10-31': ['массив дел'],

'2018-12-29': ['массив дел'],

'2019-11-26': ['массив дел'],

'2019-12-25': ['массив дел']

};

Данный объект очень удобен, если нужно будет выводить дела за определенный день, просто написав полностью ключ!

Но что делать, если нужно вывести все дела за месяц (год)?   
**Будет неудобно. Переделаем объект:**

let affairs = {

2018: {

11: {

29: ['массив дел'],

30: ['массив дел']

},

12: {

29: ['массив дел']

}

},

2019: {

10: {

31: ['массив дел']

},

11: {

26: ['массив дел']

},

12: {

25: ['массив дел'],

28: ['массив дел']

}

}

};  
Выведем все дела за 2019 год

## Преобразование в удобный вид

Так же, можно не переписывать массивы заново, а просто преобразовать их в удобный вид.  
**Например, есть массив:**

let users = [

{

id: 11,

name: 'name11',

surname: 'surname11'

},

{

id: 14,

name: 'name14',

surname: 'surname14'

},

{

id: 17,

name: 'name17',

surname: 'surname17'

}

]

**Получим из него новый массив с помощью уже изученного перебора**

for (user of users) {

convertedObj[user.id] = {

name: user.name,

surname: user.surname

}

}

**Таким образом мы получим массив такого вида**

convertedObj = {

11: {

name: 'name11',

surname: 'surname11'

},

14: {

name: 'name14',

surname: 'surname14'

},

17: {

name: 'name17',

surname: 'surname17'

}

}