2.1 Переменные let

Переменные:

Вызываем let x = 42, где x – название переменной

2.2 Числа

Литеральная запись числа. – запись любого числа

Let digit = 25;

Let fract – 25.03;

Let exp = 2e10 (20000000000);

Let expSmall – 2e-3 (0.002);

Let expFract = 1.25e3 (1250);

Let expFractSmall = 1.25e-3 (0.00125)

Let fract1 = 25.;

Let fract2 = -.25;

Для записи в двоичной системе используем:

Let bin = 0b1001

Для записи в восьмиричной системе используем:

Let bin = 0о10 (8)

Для записи в шестнадцаричной системе используем:

Let x = 0x1b; (27)

#### 2.3 Тренажёр «Переменные»

//Исправьте объявление (создание) переменных так, что бы в консоль выводились значения этих переменных, а не ошибки

/\*year = 1998;

weight let = 165;

let age = 17;

console.log(year);

console.log(weight);

console.log(age);

У нас есть переменные со значениями, необходимо исправить и вывести значения этих переменных

\*/

let year = 1998;

let weight = 165;

let age = 17;

console.log(year);

console.log(weight);

console.log(age);

#### 2.4 Математические операторы

Let x = 10 +2; // 12

Let y = 10 – 2; // 8

Let z = 10 \* 2; // 20

Let w = 10 % 3;// 1 (остаток)

Для сложных операций

Let complex = 10 + 5 \*2 – 8 / 2;

Приоритеты также как и математике (остаток приоритет такой же как и умножение и деление)

Также важное прибавляющие единицу и вычитающие единицу – энкримент и дикремент

Энкримент – прибавляющий единицу (a++)

Дикремент – вычитающий единицу (a--) , выглядит так:

Let a = 10;

a++ // (11)

a-- // (10)

перфиксная запись. Если мы выведем переменную а, то получим сначала получим старое значение 10 а только потом 11.

То есть если consol.log(a++)

То получим 10, а только потом 11.

Если хотим сразу значение,то пишем consol.log(++a) - 11

Более короткая запись операций:

Let x = 5;

X = x+5//10 или же

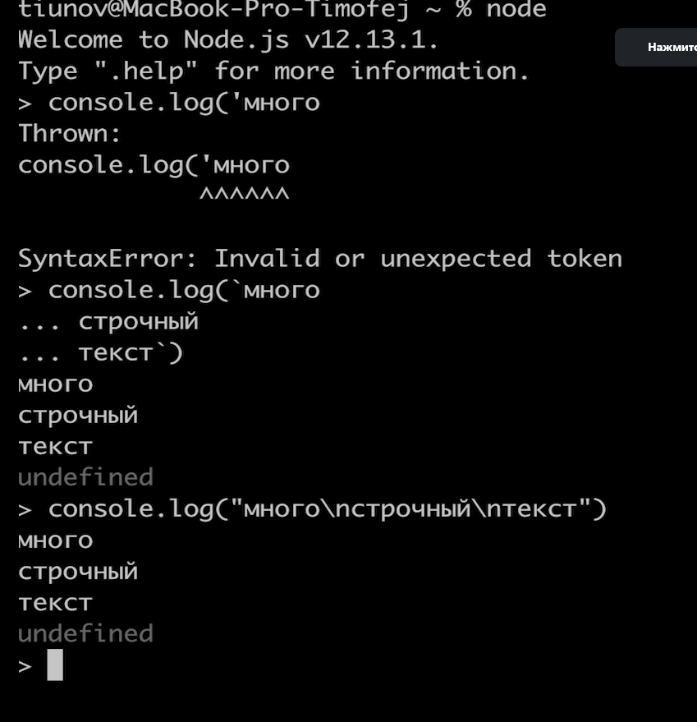
x += 5 //10

этот же прием работает на все знаки.

#### 3.1 Типы данных

#### 3.2 Строка

Перенос строки в обычных кавычках (обычные и двойные) используем \n «какое слово»\

В обратных кавычках переносы произвольные, как напишем, так и отобразится `…`.

Строки также складываются.

Значение строк в переменных пишем, и просто складываем:

Let str1 = (‘привет’)

Let str1 = (‘привет’)

Let concat = Str1 + str2.

Такой процесс называется конкатинация.

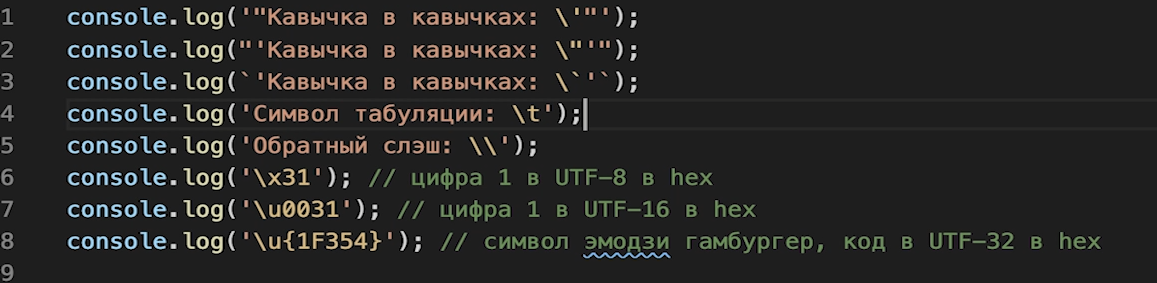
Также можно складывать и выводить вот так:

Let concat = ‘${str1}

${str2}’;

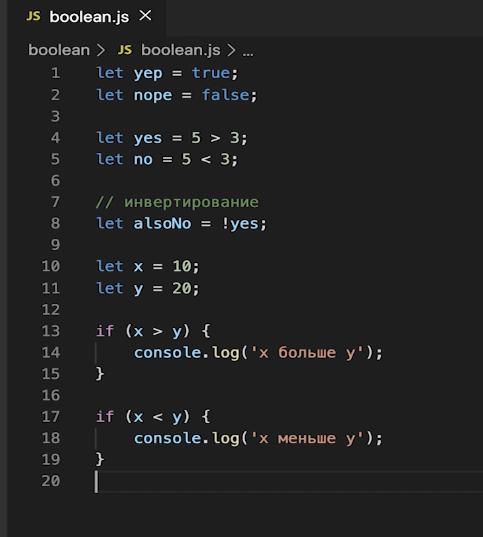
Или же

Let concat = (‘${str1}, ${str2} ! Добро пожаловать’);

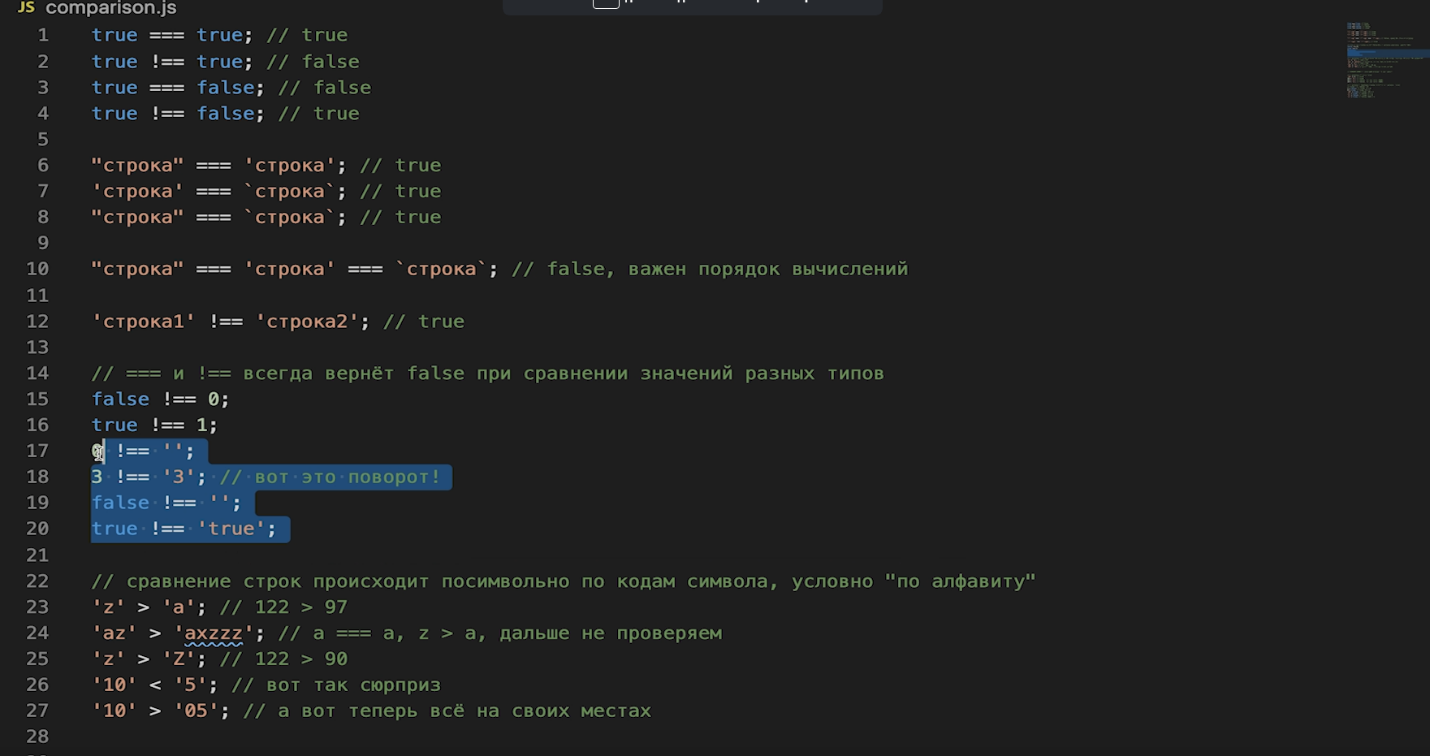


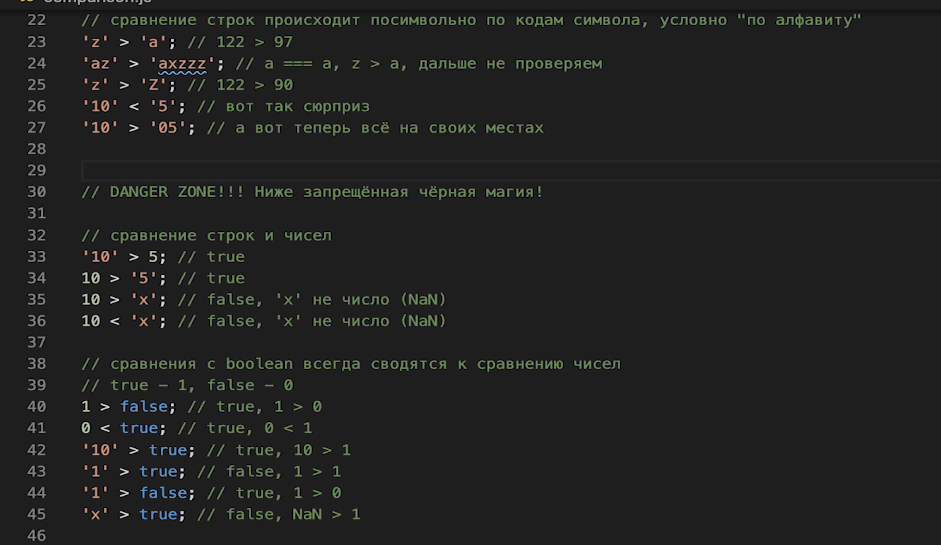
Какая то хуета. Выводим кавычки и прочую хуйню.

#### 3.4 Boolean

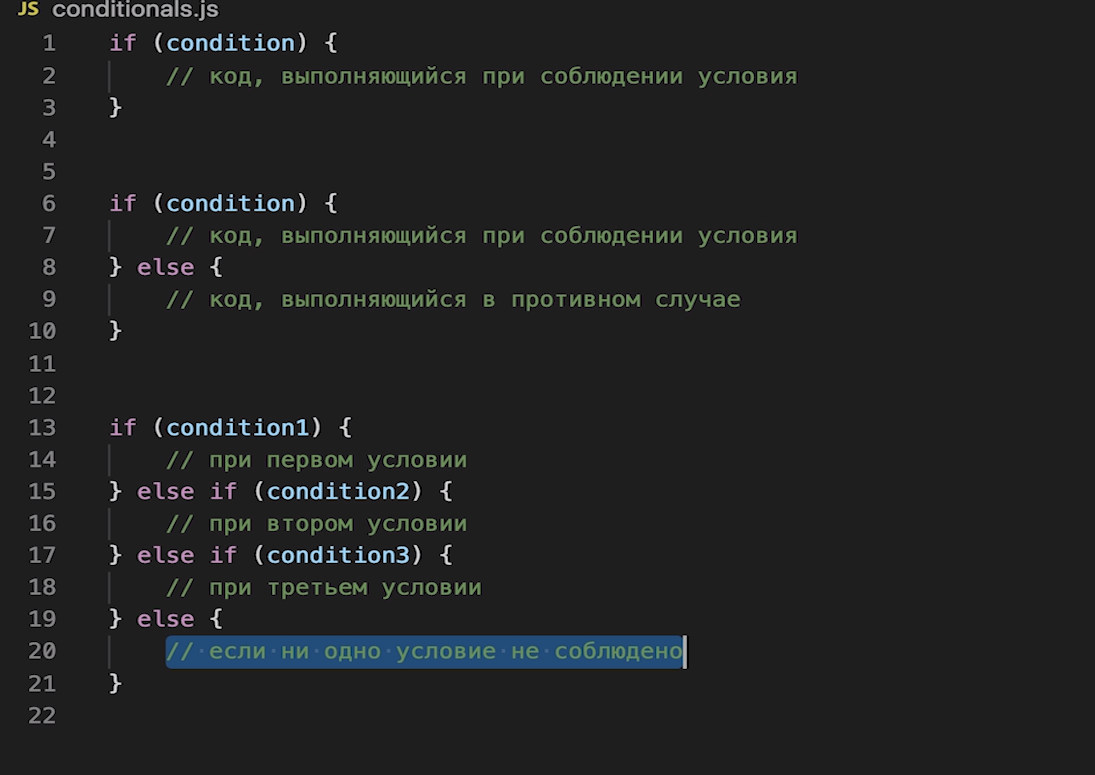


#### 3.6 Типы и сравнение





#### 3.7 Условные операторы



Наконец то дошли до условий.

**Первый вариант конструкции условия**

If (какое ни будь условие, например a>b) {

Console.log (‘а больше b’)

else{

console.log(‘b больше a’)

**Второй вариант конструкции условия**

If (какое ни будь условие, например a>b) – первое условие

Else if (второе условие){

console.log(‘b больше a’)

}

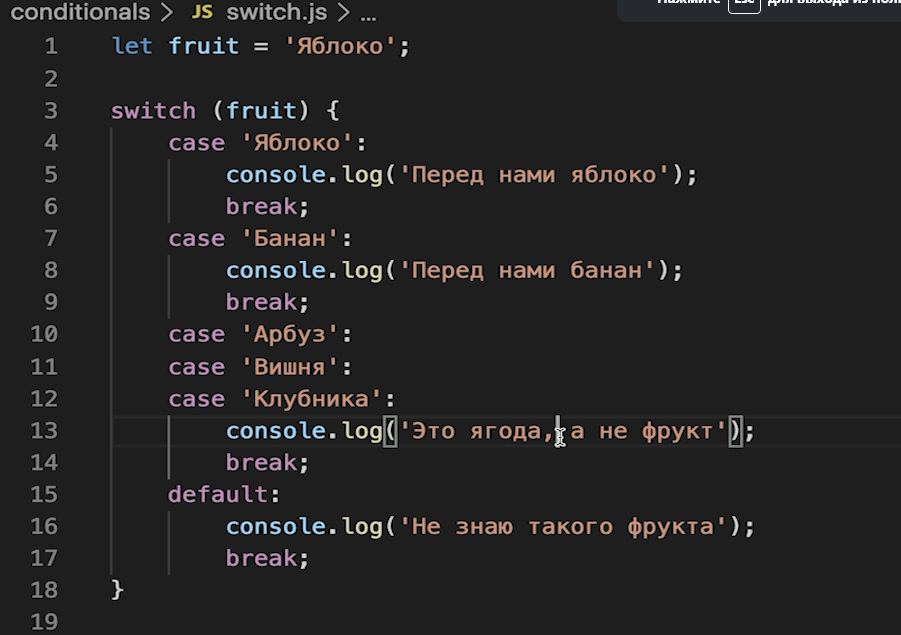
Else if (третье условие){

console.log(‘b больше a’)

}

else{если никакое условие не подошло}

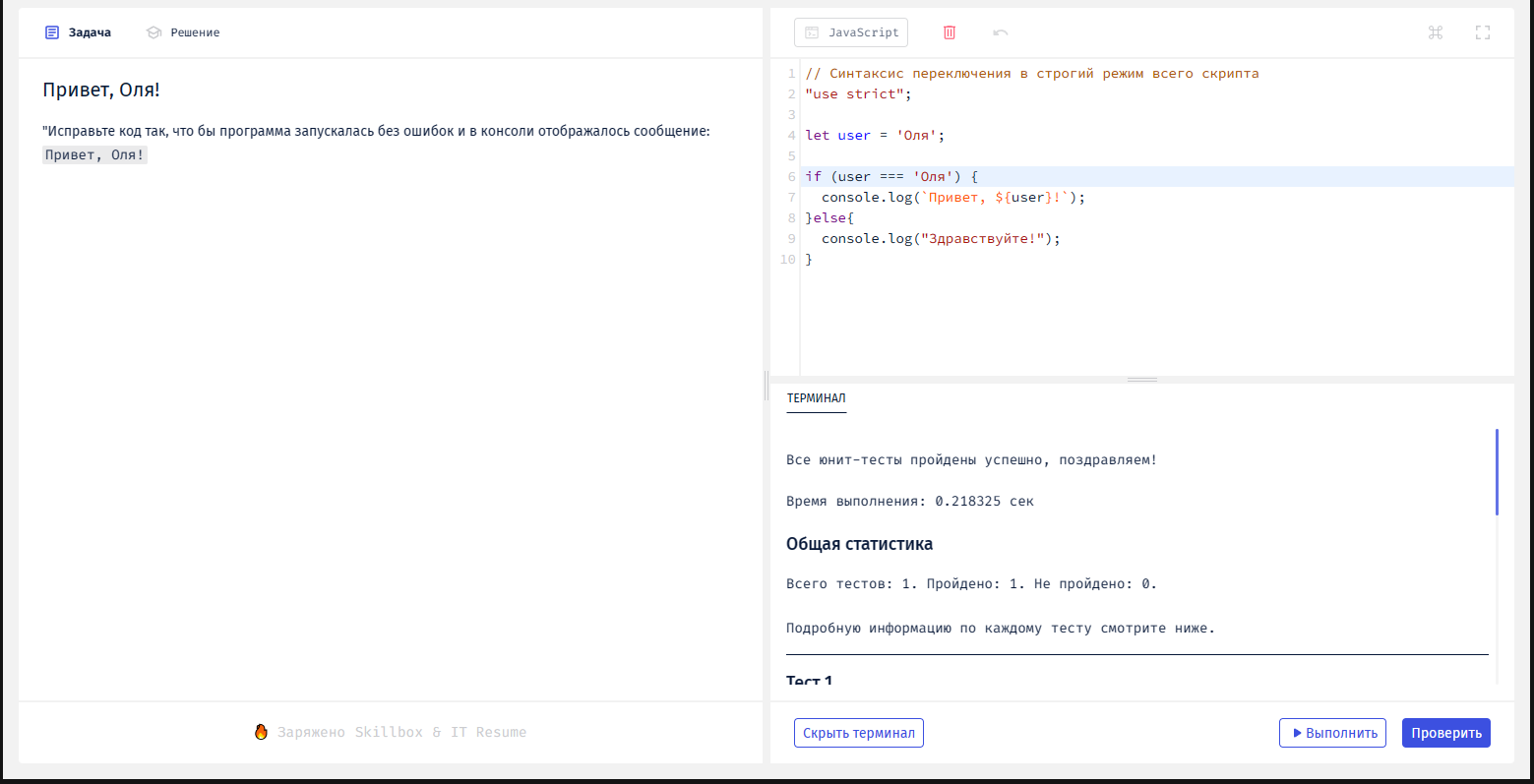
console.log(‘b больше a’)

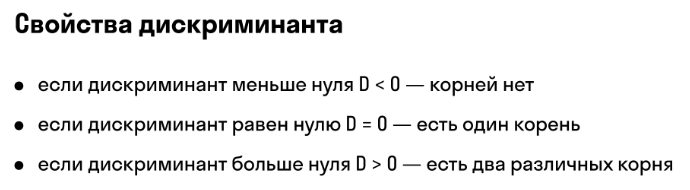


Switch – это чисто сравнения.

Есть переменная, мы ее заводим в switch и с помощью case сравниваем с и выводим какой либо результат

Если условий не много, то используем if else, если условий много, то юзаем switch. Задачка:





//Решим квадратное уравнение с помощью условий

let a = 5;

let b = 10;

let c = 3;

let d = b \* b - (4 \* a \* c);

if (d < 0){

    console.log('Дискрименант < 0, корней нет');

}

    else if (d === 0){

        let x1 = -b / 2 \* a;

        console.log('Дискрименнат = 0, один корень уравнения x1 = ', x1);

    }

    else if (d > 0){

        let dRoot = Math.sqrt(d);

        let x1 = (-b + dRoot) / (2 \* a);

        let x2 = (-b - dRoot) / (2 \* a);

        console.log(`Дискрименант равен ${dRoot}, \nесть 2 корня уравнения: \nx1 = ${x1} \nx2 = ${x2}`);

    }

#### 3.11 Тернарный оператор

Тернарный оператор – это упрощенная версия записи условия if и else.

Let x = 0;

If (math.random() > 0.5) {

X = 10;

}Else {

X= 20;

{

Тоже самое можно записать в одну строку:

X = Math.random() > 0.5 ? 10 : 20;

X = Math.random() > 0.5 – это условие

? – условный элемент

10 – значение, которое присвоится переменной при try

20 – значение переменной, которое присвоится при else

Используем только в том случае, когда нам нужно вернуть значение этой переменной!

///Тернарный оператор

//Замените конструкция if на тернарный оператор

let randomNumber = 30

console.log(`Произвольное число: ${randomNumber}`)

/\*

if (randomNumber > 20) {

  console.log("Число > 20")

} else {

  console.log("Число <= 20")

}\*/

let x = randomNumber > 20 ? 'Число > 20' : 'Число <= 20';

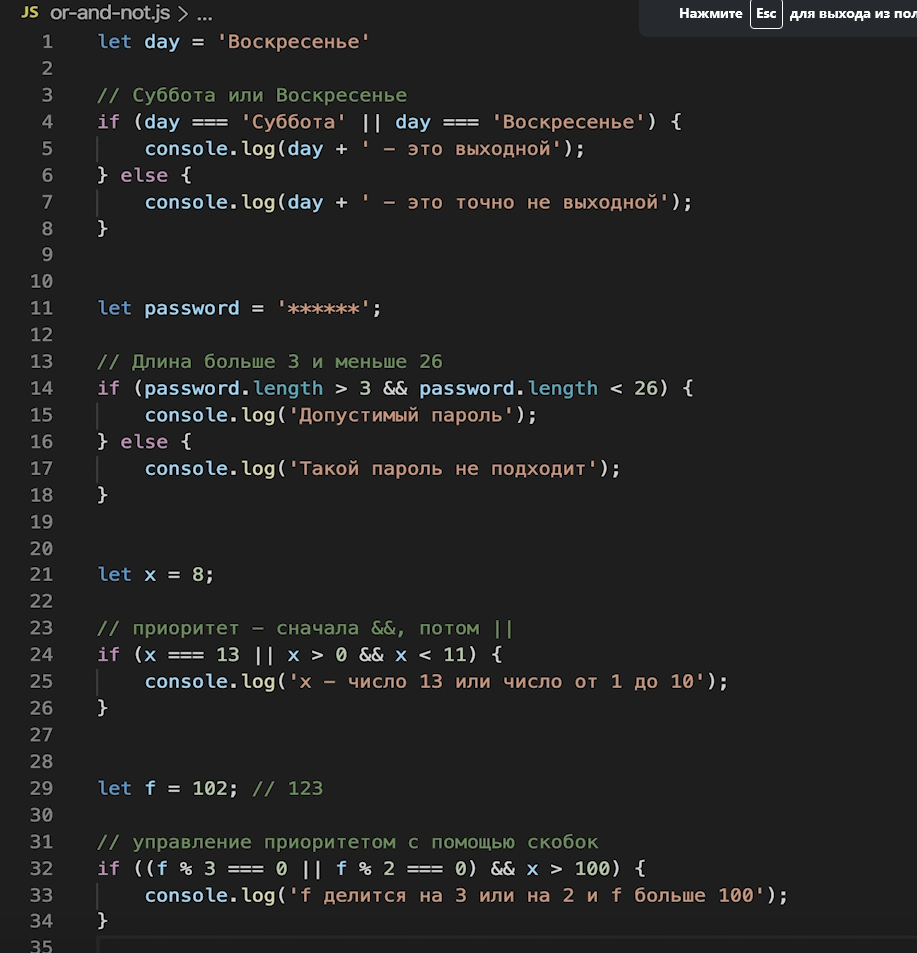
console.log(x);

#### 3.13 Основы булевой алгебры

Строится на операторах И и ИЛИ.

При чем оператор И – это &&

А оператор ИЛИ – это ||



При чем, операции сравнения выполняются в приоритете так также как и в математике, то есть, в булевой алгебре операция И выполняется в приоритете операции ИЛИ.

Также можно искусственно расставлять приоритет с помощью ()

#### 3.15 Практическая работа

#### Задача 1

##### **Цель задания**

Научиться искать подстроку внутри строки и попрактиковаться со сложными условиями.

##### **Что нужно сделать**

В переменную password запишите строку с любым произвольным паролем. Проверьте надёжность пароля с помощью условного оператора if. Пароль является надёжным, когда в нём есть хотя бы четыре символа, один из которых — это дефис или нижнее подчёркивание. Выведите в консоль сообщения «Пароль надёжный» или «Пароль недостаточно надёжный».

##### **Советы и рекомендации**

Для проверки наличия в строке другой строки можно воспользоваться конструкцией password.includes('x'), где 'x' — строка для поиска. Поэкспериментируйте с этой командой, посмотрите, что она будет выводить в консоль, попробуйте подставить разные параметры. Это поможет понять принцип её работы.

##### **Проверка результата**

Для проверки запустите код с разными вариантами надёжных и ненадёжных паролей. Примеры корректных паролей:

* 1234-
* 4321\_
* qaz-xsw
* \_zxd

Примеры некорректных паролей:

* \_-a
* qaz
* \_-3
* 123456789

##### **Что оценивается**

Код корректно выводит сообщение в зависимости от значения переменной password.

#### Задача 2

##### **Цель задания**

Узнать, как преобразовывать строку в верхний/нижний регистр и извлекать произвольные куски из строки.

##### **Что нужно сделать**

В переменных userName, userSurname даны имя и фамилия пользователя. При этом в строках беспорядок с большими и маленькими буквами, и нужно оформить строки единообразно. Для этого первые буквы имени и фамилии приведите к верхнему регистру (большие буквы), а оставшиеся — к нижнему (маленькие буквы). Запишите результат в новые переменные и выведите их значения с помощью console.log. С помощью тернарных операторов и console.log выведите сообщение «Имя было преобразовано» или «Имя осталось без изменений» для имени и фамилии в зависимости от того, были ли исходные строки равны преобразованным.

##### **Проверка результата**

Для любых имени и фамилии в любом регистре должны выводиться такие же имя и фамилия, но первая буква у них большая, а все остальные — маленькие.

##### **Что оценивается**

Код корректно выводит все сообщения в зависимости от значения переменных userName и userSurname.

##### **Советы и рекомендации**

Для получения куска строки можно воспользоваться конструкцией str.substring(indexA, indexB), где str — название переменной с исходной строкой, indexA — целое число от нуля до длины строки, указывающее номер символа начала куска строки, indexB — целое число от нуля до длины строки, указывающее номер символа конца куска строки. Если indexB не указывать, то вы получите кусок от indexA до конца строки. Например, так можно получить первый символ строки: let first = str.substring(0, 1), а так — остаток строки: let last = str.substring(1). Конструкцию можно присвоить переменной, с которой потом можно работать как с обычной строкой. Для преобразования букв строки к верхнему регистру воспользуйтесь конструкцией str.toUpperCase(), к нижнему — str.toLowerCase(). Результат аналогично можно присвоить переменной.

#### Задание 3

##### **Цель задания**

Научиться проверять числа на чётность.

##### **Что нужно сделать**

В переменной number записано число. Необходимо с помощью console.log вывести сообщение, указывающее на чётность или нечётность числа.

##### **Проверка результата**

Для проверки подставляйте различные значения в переменную number и оценивайте корректность результата.

Пример:

* number = 2, console.log(“Число чётное”);
* number = 5, console.log(“Число нечётное”);
* number = 8, console.log(“Число чётное”);

##### **Что оценивается**

Алгоритм выводит правильный ответ на экран.

##### **Советы и рекомендации**

Для проверки числа на чётность можно использовать оператор «Остаток от деления» — %. Например, если произвести операцию 5 % 2, вы получаете 1. Это означает, что число 5 — нечётное. Если использовать другое число: 4 % 2 , в ответе получается 0. Это означает, что число чётное. Попробуйте совместить эту команду с условным оператором.

Сейчас может быть непонятно, как применять эти задачи в реальной практике. Но после изучения базовых команд — инструментов языка программирования мы перейдём к более реальным заданиям и работе с графическими элементами на странице в браузере.

console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

console.log('Задача №1 - Пароль')

let password = 'sdc2sa\_dfd';

if ((password.length >= 4) && ((password.includes('\_')) || (password.includes('-')))){

    console.log(password + '- Пароль надежный')

}

else{

    console.log(password + '- Пароль ненадежный');

}

console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

console.log('Задача №2 - Имя пользователя')

let userName = 'даРья';

let userSurname = 'сМануховА';

let fersuserName = userName.substring(0, 1);

let lastuserName = userName.substring(1);

let ferstuserSurname = userSurname.substring(0, 1);

let lastuserSurname = userSurname.substring(1);

console.log(fersuserName.toUpperCase() + lastuserName.toLowerCase(), ferstuserSurname.toUpperCase() + lastuserSurname.toLowerCase());

console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

console.log('Задача №3 - числа на четность')

let number = 5;

let ziro = (number === 0) ? ('- Число 0 на 2 не делится') : ( 'Остаток от деления  ' + number % 2);

if ((ziro === 0) || (number === 0)){

    console.log(number + ' - Число четное')

}

else{

    console.log(number + ' - Число не четное')

}

#### 4.1 Примитивные типы и массивы

Сложные типы данных – это данные включающие все в себя несколько подтипов типов данных

#### 4.2 Что такое массив

Массив – сложный тип данных, который позволяет хранить множество разных данных в определенном порядке

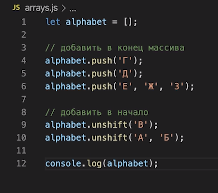
Let fruit = [‘яблоко’, ‘груша’, ‘апельсин’]; - массив со строками.

Элементы расположены в определенном порядке и каждого есть свой индекс, начиная с лева на право с 0. То есть [‘яблоко – (0)’, ‘груша – (1)’, ‘апельсин – (2)’]

Через консоль можно обращаться к элементам массива:

Let fruit = [‘яблоко’, ‘груша’, ‘апельсин’];

Fruit[0] – это будет яблоко

Если мы не знаем колличество элементов массива, создаем пустой и добавляем туда по готовности элементы с помощью push – в конец, unshift – в начало:

Let aphavit = [];

Aphavit.push(‘г’)

Aphavit. Unshift(‘Д’) и тд

Также можем узнать длину массива с помощью команды length.

Let fruit = [‘яблоко’, ‘груша’, ‘апельсин’];

Fruit.length – выведет 3 элемента масиива.

С помощью length можем вывести последний элемент массива:

Fruit[Fruit.length – 1] – выведет апельсин.

Также с помощью индекса можем изменять элементы массива, если допустили ошибку с помощью greeting[индекс массива]:

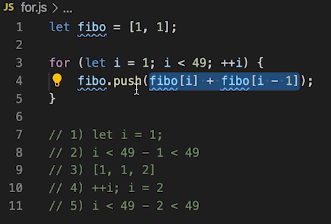
Let fruit = [‘яблоко’, ‘груша’, ‘апельсин’];

Fruit[2] = ‘арбуз’

Console.log(fruit[2]); - выведет исправленный массив с арбузом.

#### 4.3 Цикл for

Цикл for позволяет выполнять определённые действия n количество раз

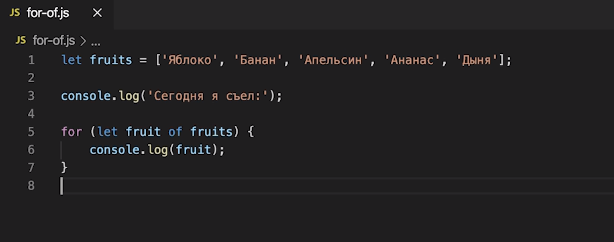




#### 4.4 Циклы for of, for in

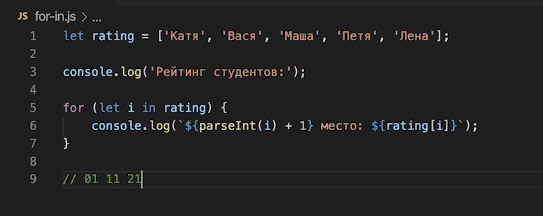
**For of**

Короче



В строке №5 переменной fruit поочередно присваивается элемент массива. При каждом выполнении цикла, разный фрукт в переменной fruit.

**Цикл for in**



Короче, в переменную i заносится строка (она же одновременно и индекс)

В данном примере сопоставляется имя студента и номер его посадочного места.

В строке №6 мы выводим значение первого элемента массива преобразованного в помощью parseint в число (0 +1) получаем посадочное место, далее выводим само значение строки и получаем: 1 место: Катя

#### 4.5 Циклы while и do-while



Пока не выполнится условие, цикл будет повторяться, уменьшая переменную на 10.

//цикл while

let varior = 500;

console.log('Вариативный объект', + varior + 'шт.')

while (varior > 0){

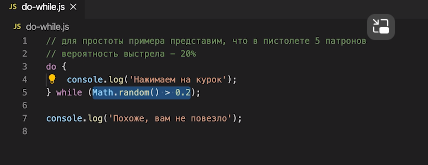
    varior = varior - 50;

    console.log('первая итерация', + varior + ' шт');

}

console.log('цикл завершен');

**цикл do white**

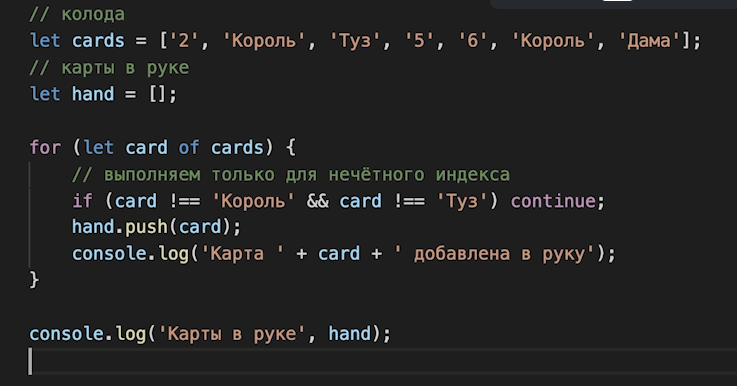
**То же самое, что и while, за исключением, что условие проверяется после выполнения цикла а не в начале.**

Если выстрел произошел, вам не повезло. Тот же самый принцип, что и while, только не забываем про do в начале.

***Используем только в том случае, когда мы уверены, что тело цикла должно выполниться хотя бы 1 раз.***

#### 4.6 Операторы continue и break

**continue**



Операция continue – нужна для продолжения выполнения цикла в том случае, который описан выше., как например с колодой карт. Если нам из списка массива не выпадает то, что нам нужно, мы снова проходимся по нему.

С помощью **for fo** мы пробегаемся по нашему массиву и записываем в переменную card каждый элемент массива после каждой итерации и далее с помощью if сравниваем его с условием.

**break**

let cards = ['2', 'Туз', 'Король', 'Дама', 'шестерка', 'семерка'];

let found = false;

for (let card of cards){

    console.log(`вытащили карту ${cards}`);

    if(card === 'Король'){

        found = true;

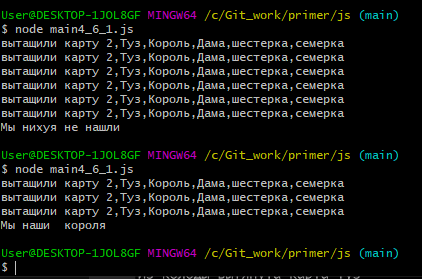
        break;

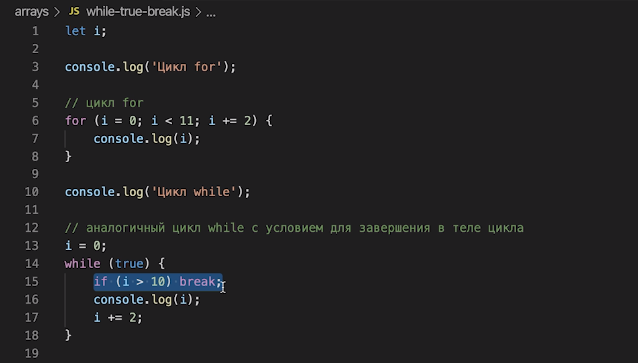
    }

}

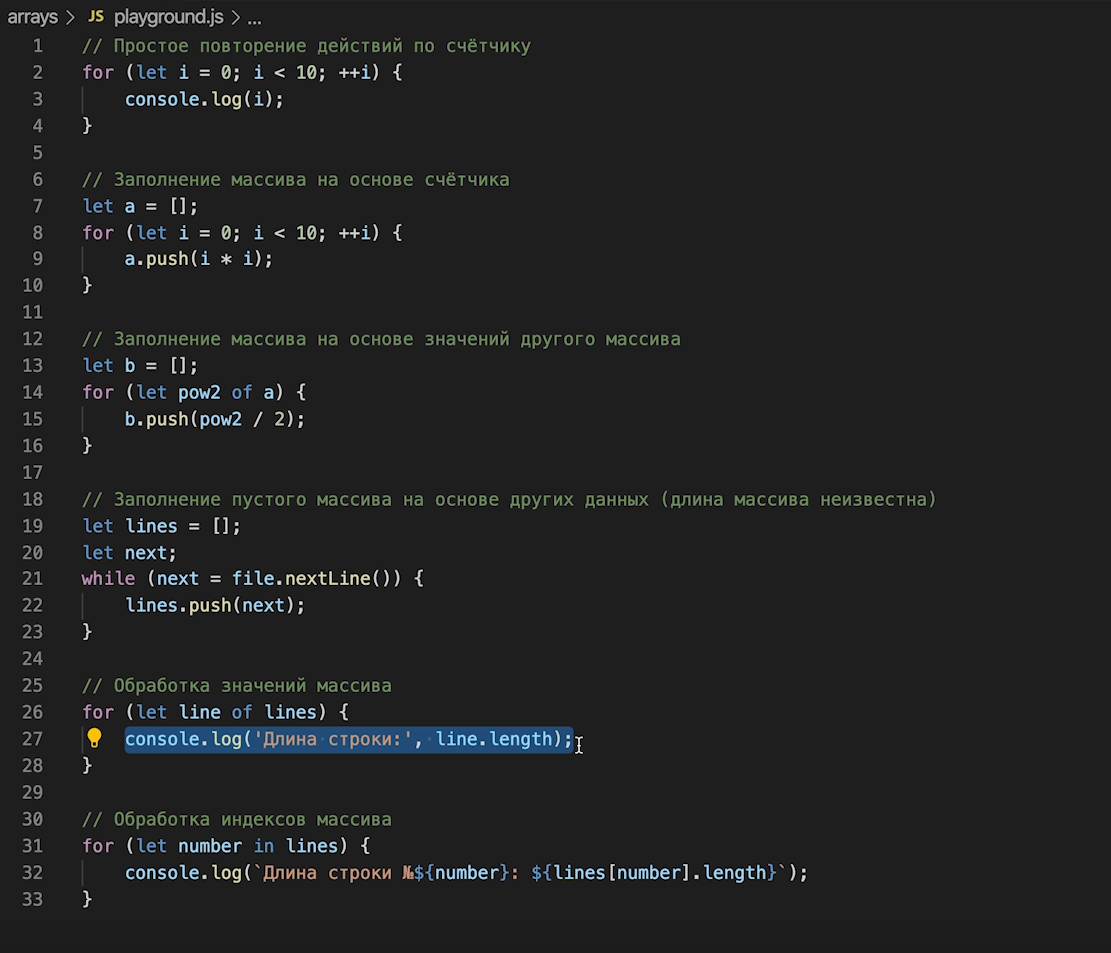
console.log(found ? 'Мы наши  короля' : 'Мы нихуя не нашли');

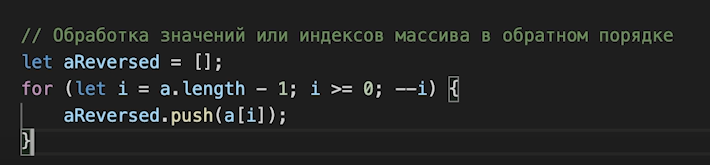
**Не понимаю, почему у меня в переменную card выводит все элементы массива а не поочередно каждый**

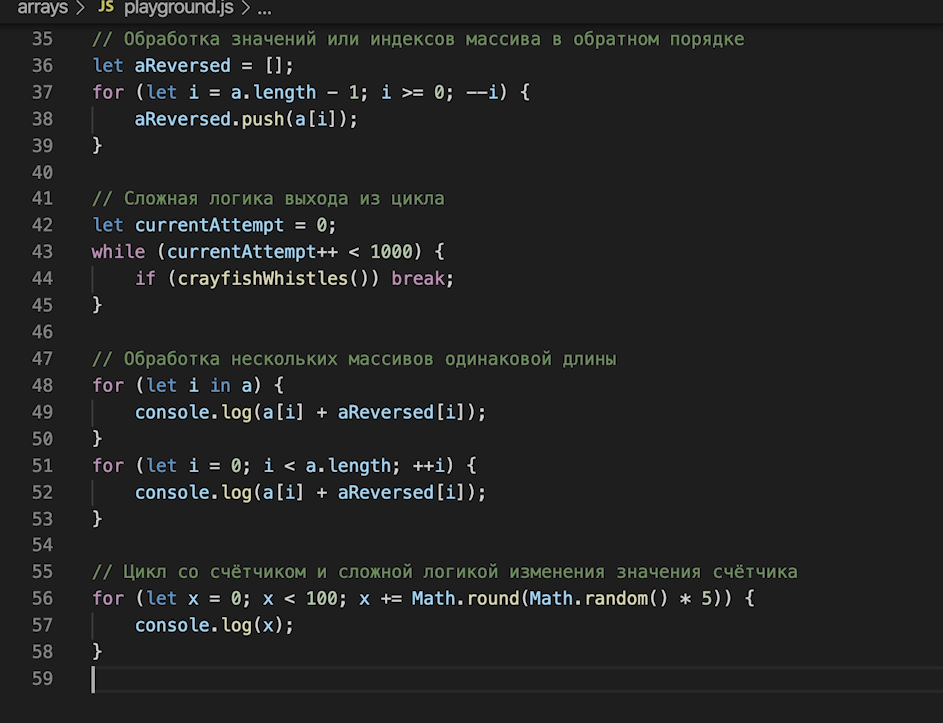




#### 4.7 Какой цикл выбрать







#### 4.8 Перемешивание массива

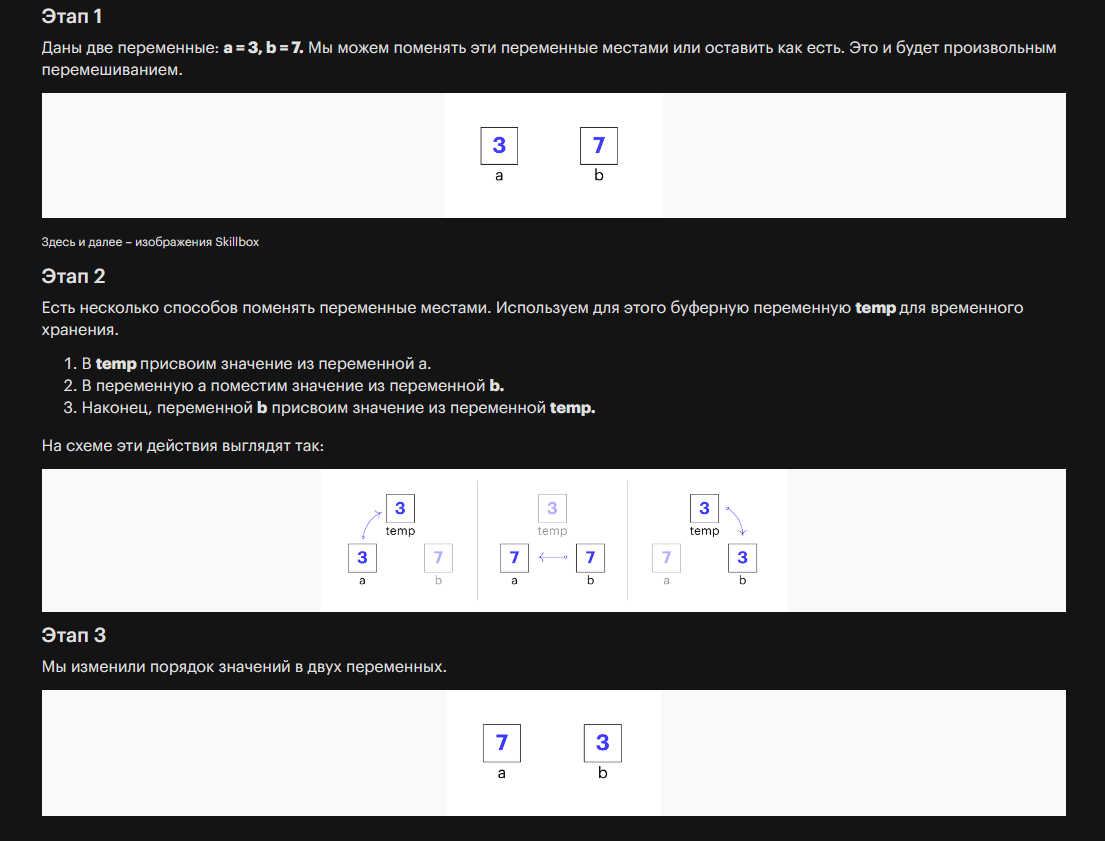
#### Что такое перемешивание и зачем это нужно

Перемешивание массива — это перемещение элементов в произвольном порядке. Процесс происходит при изменении индексов у элементов массива.

Операция перемешивания используется для широкого круга задач. Например, при создании игры в карты нужно задать функцию перемешивания колоды карт, то есть упорядоченного массива.

#### Перемешивание двух переменных

Произвольное перемешивание — тип перемешивания, при котором порядок выбирается случайным образом. В начале разберём простейшее перемешивание двух переменных с числами.



Переменная temp – буферная переменная.

### **Перемешивание элементов массива**

Самый популярный и простой алгоритм перемешивания элементов массива — «тасование Фишера — Йетса».

Он заключается в следующем:

* цикл проходит по каждому элементу массива;
* для каждого следующего элемента находит произвольную пару (рандомный индекс элемента);
* после этого алгоритм меняет элементы местами.

Рассмотрим этот алгоритм более подробно на примере.

##### **Шаг 1**

Дан массив: **arr = [1,2,3,4,5,6,7,8,9,10].** Длина этого массива равна 10. Соответственно, и количество повторений цикла будет равно 10.

##### **Шаг 2**

Создадим цикл со счётчиком **i**, который будет перебирать каждый элемент массива **arr**по очереди. Назовём каждый последующий элемент **arr[i]**.

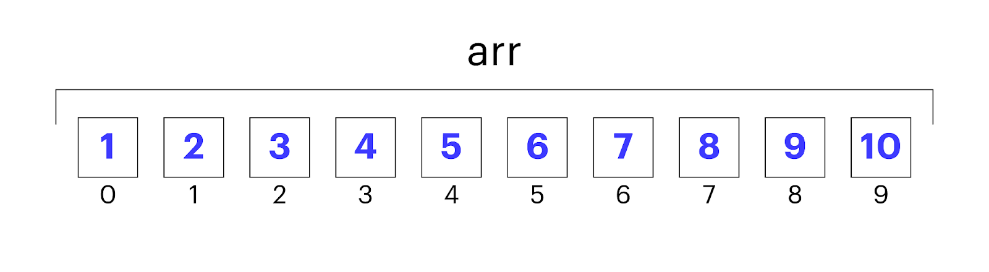
##### **Шаг 3**

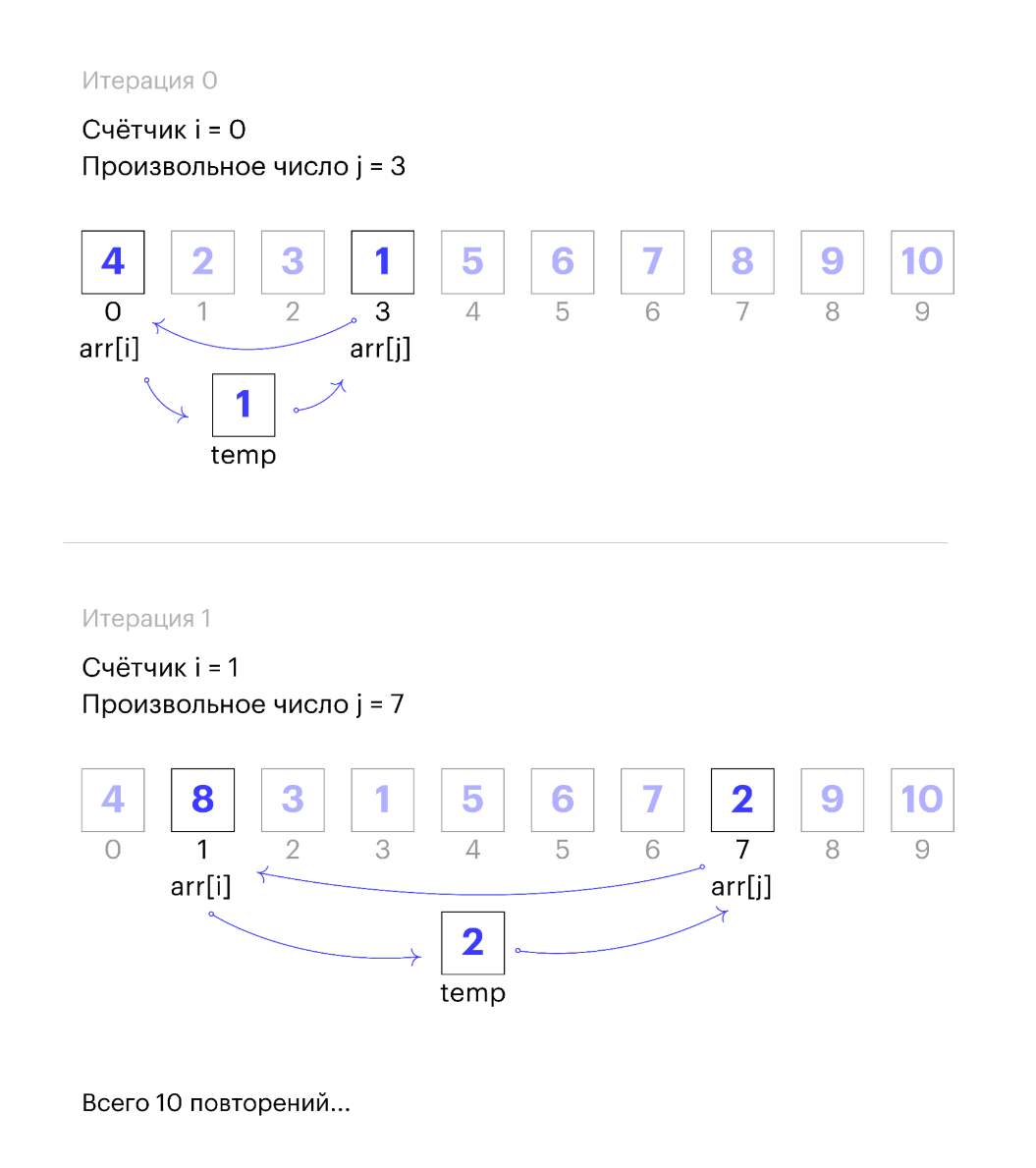
Внутри тела цикла при каждом повторении сгенерируем произвольное число в диапазоне от 0 до длины массива, то есть от 0 до 9, так как длина массива arr равна 10. Полученное число**j**будет индексом произвольного элемента **arr[j].**

##### **Шаг 4**

В итоге мы получим два элемента внутри цикла — **arr[i]** и **arr[j]**. По аналогии с перемешиванием двух переменных мы можем поменять местами эти элементы с использованием буферной переменной **temp**.

##### **Шаг 5**

Далее цикл повторяет эти действия, пока весь массив не будет перемешан в произвольном порядке. Пример работы этого алгоритма показан на блок-схеме:



#### 4.9 Поиск в массиве

### **Определение поиска в массиве**

Поиском в массиве обычно называют определение порядкового номера элемента (индекса) в массиве. Например, есть задача узнать индекс имени **‘Ян’** в массиве имён **[‘Аня’,‘Саша’,‘Ян’,‘Никита’,‘Вика’,‘Олег’]**. Индексом имени**‘Ян’** будет **2**, так как индексы в массиве начинаются с нуля. Давайте разберём, как реализовать поиск в программе.

### **Простой поиск элемента в массиве**

##### **Шаг 1**

Дан массив имён студентов: **students = [‘Валя’,‘Игорь’,‘Витя’,‘Женя’,‘Таня’,‘Рома’]**.

Есть задача найти порядковый номер имени **Женя**в этом массиве.

##### **Шаг 2**

Для начала создадим переменную **index = −1**. Если в результате выполнения программы в **index** останется отрицательное число, значит, что элемент со значением **‘Женя’** в массиве не был найден.

Изображение: Skillbox

##### **Шаг 3**

Далее с помощью классического цикла со счётчиком **i**проходим по каждому элементу массива и, используя условный оператор **if,** проверяем, равно ли значение элемента массива значению **‘Женя’.**Если да, присваиваем переменной**index** счётчик цикла **i** и завершаем работу цикла. Это и будет порядковый номер найденного элемента.  
Цикл будет повторять проверку до тех пор, пока программа не найдёт нужный элемент в массиве или цикл не закончится.

### **Итог**

Вы научились выполнять простейший поиск элементов в массиве — задача, которая часто встречается в практике.

Хотя в JS есть встроенные функции для поиска индекса элемента, рекомендуем попробовать написать код без них, чтобы лучше освоить навыки в обработке массивов.

В этом материале мы разобрали только алгоритм для выполнения поиска элементов. Теперь вы можете самостоятельно написать код программы, чтобы потренировать навыки программирования.

В следующих блоках вы познакомитесь с другими интересными приёмами программирования.



Задача 1

Цель задачи

Научиться генерировать произвольные массивы и адаптировать существующий код под ситуацию.

Что нужно сделать

Напишите генератор массивов длиной count со случайными числами от n до m. Учтите, что n и m могут быть отрицательными, а также может быть n > m или n < m.

Выведите результат с помощью console.log.

Проверка результата

Для проверки подставляйте различные значения count, m, n и смотрите на корректность результата. Примеры значений для проверки:

n = 0, m = 100, count = 100; вывод результата в консоль: [2, 51, 47, … , 95 ];

n = 2, m = 5, count = 50; вывод результата в консоль: [2, 5, 3, … , 2 ];

n = 100, m = −5, count = 70; вывод результата в консоль: [−2, 35, 94, … , −4 ];

n = −3, m = −10, count = 42; вывод результата в консоль: [−4, −3, −9, … , −7 ].

Критерии оценки

Для count, m, n код генерирует соответствующий массив.

Рекомендации к выполнению

Попробуйте использовать несколько видов циклов и выбрать тот, который вы считаете наиболее подходящим для решения задачи.

Задача 2

Цель задачи

Научиться перемешивать массив чисел.

Что нужно сделать

Создайте с помощью цикла for массив упорядоченных чисел с количеством чисел, равным count. Например:

count = 5; соответствует массив [1,2,3,4,5];

count = 7; соответствует массив [1,2,3,4,5,6,7];

count = 3; соответствует массив [1,2,3].

С помощью второго цикла перемешайте этот массив.

Выведите получившийся результат на экран с помощью console.log.

Проверка результата

Для любых значений count вы должны получать новый массив с заданной длиной. Например:

count = 5; пример результата вывода в консоль: [2,5,1,3,4];

count = 7; пример результата вывода в консоль: [5,1,3,2,7,6,4];

count = 3; пример результата вывода в консоль: [2,1,3].

Критерии оценки

Код выводит на экран массив с заданной длиной.

Рекомендации к выполнению

Надёжно перемешать массив вы сможете с помощью так называемого алгоритма Фишера — Йетса (Fisher — Yates). С помощью цикла вы получаете индекс (порядковый номер) каждого элемента массива по очереди. После получения индекса i вам нужно сгенерировать индекс произвольного элемента j. Генерировать произвольные числа от 0 до count вы уже умеете. Так вы получите два индекса элементов. Теперь остаётся поменять их местами. Чтобы поменять местами два элемента массива, вы можете использовать следующий способ со вспомогательной переменной temp:

temp = myArray[i];

myArray[i] = myArray[j];

myArray[j] = temp;

Все действия с получением произвольного числа j и перестановка чисел выполняются внутри тела цикла.

Более подробную информацию об алгоритме перемешивания вы без труда сможете найти в любом поисковике.

Задача 3

Цель задачи

Научиться искать индекс (порядковый номер) нужного элемента в массиве.

Что нужно сделать

С помощью цикла найдите индекс (порядковый номер) элемента массива из предыдущего задания с числом n. Если такой элемент не будет найден, то выведите соответствующее сообщение на экран.

Проверка результата

Для любых значений массива из второго задания и числа n вы должны получать индекс элемента в массиве или сообщение о том, что такого элемента нет.

Например:

array = [2,5,1,3,4], n = 3; индекс элемента = 3;

array = [5,1,3,2,7,6,4], n = 1; индекс элемента = 1;

array = [2,1,3], n = 7; элемент не найден.

Критерии оценки

Код выводит на экран порядковый номер элемента массива или сообщение о том, что элемент не найден.

Задача 4

Цель задания

Научиться обрабатывать массивы одним циклом.

Что нужно сделать

Даны два массива:

arr1 = [2, 2, 17, 21, 45, 12, 54, 31, 53]

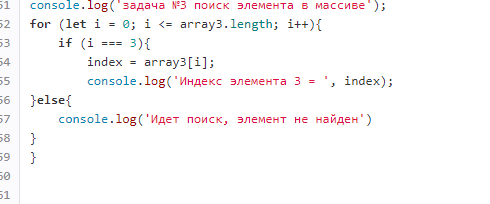
arr2 = [12, 44, 23, 5]

Напишите программу, которая будет объединять два массива: arr1 и arr2. Результат объединения нужно вывести в консоль с помощью команды console.log в таком виде:

[2, 2, 17, 21, 45, 12, 54, 31, 53, 12, 44, 23, 5]

Ринат, добрый день! ☀️☀️

✅    в данном случае если элемент не найден



буедт выводиться несколько раз что не правильно.

let n = 3;

let index = -1;

потом используем ваш цикл и i сравниваем с n

внутри условия в index записывает i

А дальше после цикла мы проверяем, если index равно -1 значит элемент не найден. в противном случае элемент найден

✅   для более верного вычисления которое включало бы минимальное и максимальное число, я бы расширил формулу:

https://go.skillbox.ru/media/files/c40a1cad-fd38-43cc-b146-db6c5973a957/1714026785342.png

Math.floor( Math.random() \* Math.abs(m - n + 1)) + Math.min(n, m);

#### 5.1 Интро: Переиспользование кода

Ранее мы уже сталкивались с функциями console Math и тд

#### 5.2 Базовый синтаксис

Есть название функции, есть тело функции

Function functionDay{ - название функции

Console.log (‘вызов функции’) – Тело функции

}

Чтобы вызвать функцию необходимо написать название функции и (): functionDay().

Также можно объявить функцию через переменную, присвоив ей значение функции, тогда название функции можно будет опустить, а вызывать не по имени функции, а по имени переменной используя () в конце.

Разница в том, что вызов функции так как он есть, не через переменную – ее можно будет вызывать глобально, если функцию объявить через переменную, вызвать до ее объявления мы не сможем.

#### 5.2 Базовый синтаксис

Аргументы функции – параметры с которыми работает функция.

Function greet(who) {

Console.log(`Привет, ${who}`)

}

Greet(‘Петя’);

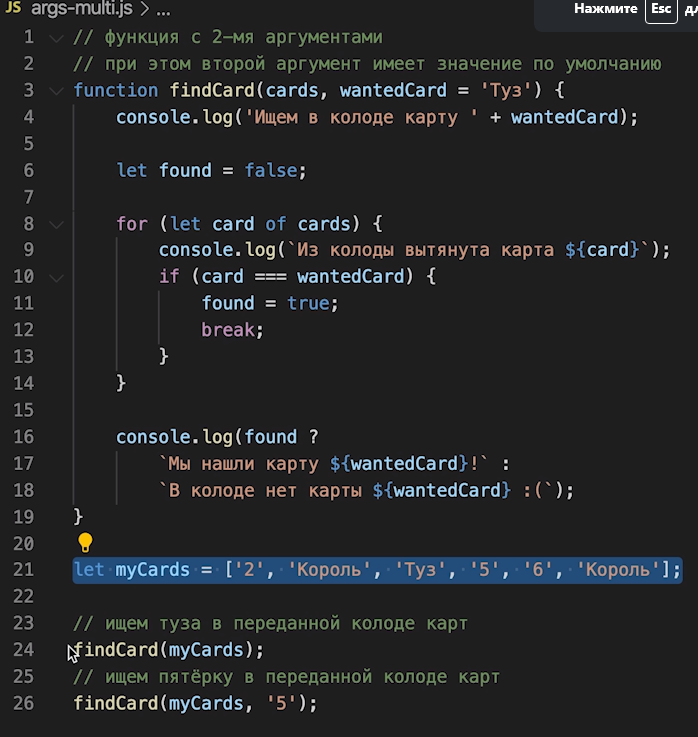
Greet – имя функции

(who) – аргумент функции

Аргументы функции можно использовать также как и переменные, но только внутри функции

Также аргументу функции можно присвоить какое значение по умолчанию

Function greet(who = ‘незнакомец’)



Cards – массив с картами, wantedCard – искомая карта.

Присваиваем card значение массива Mycards поочередно, сравниваем и выводим.

При вызове функции можем задать любое искомое значение, которое нам необходимо найти в массиве: findCard(myCards, ‘5’);

function findCard(index, wantedCard = 'volkswagen') {

    console.log('Ищем в колоде карту' + wantedCard);

    let found = false;

    for (let card of index){

        console.log(`Из колоды вытянута карта ${card}`);

        if (card === wantedCard){

            found = true;

            break;

        }

    }

    console.log(found ? `Мы нашли карту ${wantedCard}!` : `В колоде нет карты ${wantedCard}:( `);

}

let cars = ['mazda', 'mers', 'volkswagen', 'porh', 'kia'];

findCard(cars);

Еще раз!

function findCard(index, wantedCard = 'volkswagen') {

}

Index – это свободный параметр, обзови его как хочешь, туда подставляется какой либо тип данных, массив или еще что то.

wantedCard = 'volkswagen' – второй параметр, которому присвоено какое то значение.

let found = false; - заведомо задаем какое значение правды или лжи, что бы потом к нему прицепится при следующем цикле.

for (let card of index){

console.log(`Из колоды вытянута карта ${card}`);

if (card === wantedCard){

found = true;

break;

}

}

card – произвольная переменная куда будут заносится наши данные из index, с которыми будем работать. Повторяю, index может быть любой переменной, будь то массив или еще что то.

Далее если значение массива, которое поочередно заносится в переменную card будет равно нашему второму параметру, значение которого нам задано, то как раз переменная found будет уже не false, а true. И тормозим функцию.

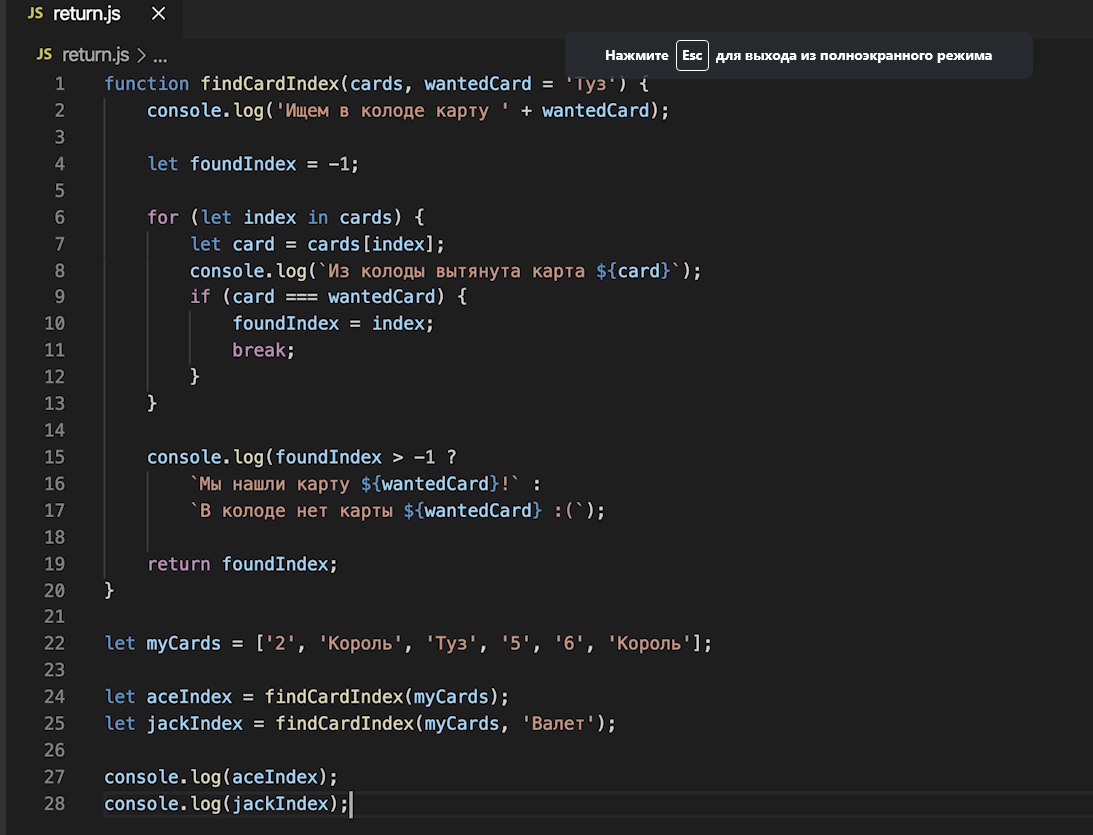
console.log(found ? `Мы нашли карту ${wantedCard}!` : `В колоде нет карты ${wantedCard}:( `);

далее упрощенная форма записи цикла if.

То есть: если found у нас true, то Мы нашли карту ${wantedCard}, если false, то В колоде нет карты ${wantedCard.

Вроде разобрались.

#### 5.4 Возвращаемое значение и undefined



Возвращаем значение функции (ее результат) с помощью return.

Отвечу как мировой эксперт в программировании с наградой "Лучший разработчик Java

Код, который вы привели, почти работает правильно, однако есть несколько моментов, которые нужно исправить:

1. В теле цикла вы повторно объявляете переменную card с помощью let, что может привести к конфликту с аргументом функции card. Название переменной в цикле лучше заменить на другое.
2. В первой строке функции в выводе не хватает пробела после слова "колоде" перед переменной wantedCard.
3. В цикле for...in индекс массива будет строкой, что может повлиять на правильность работы программы (например, в случае строгого сравнения типов).

Ниже исправленный код:

javascript

Копировать код

function findCardIndex(cards, wantedCard = 'Туз') {

console.log('Ищем карту в колоде ' + wantedCard);

let foundIndex = -1;

// Используем for...of для работы с элементами массива

for (let index = 0; index < cards.length; index++) {

let currentCard = cards[index];

console.log(`Из колоды вытянута карта ${currentCard}`);

if (currentCard === wantedCard) {

foundIndex = index;

break;

}

}

console.log(

foundIndex > -1

? `Мы нашли карту ${wantedCard}`

: `В колоде нет карты ${wantedCard}`

);

return foundIndex;

}

let myCard = ['2', 'Король', 'Туз', '5', '6', 'Король'];

let aceIndex = findCardIndex(myCard);

let jackIndex = findCardIndex(myCard, 'Валет');

**Изменения:**

1. Переименовал переменную card на currentCard внутри цикла, чтобы избежать конфликтов с аргументами.
2. Заменил цикл for...in на for для правильного доступа к индексам массива, так как цикл for...in используется для перечисления свойств объекта, а не для итерации по элементам массива.
3. Добавил пробел в выводе строки "Ищем карту в колоде".

4o

##### Вы сказали:

let currentCard = cards[index]; объясни эту строку

###### ChatGPT сказал:

ChatGPT

Отвечу как ведущий эксперт по программированию с наградой "Лучший преподаватель по JavaScript" Российской академии наук.

Строка let currentCard = cards[index]; выполняет следующие действия:

1. **cards[index]**: Это обращение к элементу массива cards по его индексу. Массив — это структура данных, в которой элементы нумеруются от 0 и выше. В переменной index хранится текущий индекс в процессе итерации по массиву с помощью цикла for. Например, если index равен 2, то cards[index] вернет третий элемент массива cards.
2. **let currentCard**: ЭтаcurrentCard, которая будет хранить значение элем"Туз", то переменная currentCard будет равна "Туз" на данной итерации.
3. **Назначен**ие значения: Строкаlet currentCard = cards[index]; берет элемент массива с индексом index и присваивает егоcurrentCard. Это позволяет вам использовать значение этого элемента для выполнения дальнейших операций внутри цикла, например, для сравнения, вывода в консоль и т.д.

Пример:

javascript

Копировать код

let cards = ['2', 'Король', 'Туз', '5'];

let index = 2;

let currentCard = cards[index]; // currentCard будет равно 'Туз'

console.log(currentCard); // Выведет: Туз

Таким образом, эта строка просто извлекает элемент массива на текущей итерации и сохраняет его в переменной для дальнейшего использования.

4o

##### Вы сказали:

return foundIndex; объясни это

###### ChatGPT сказал:

ChatGPT

Отвечу как мировой эксперт в программировании с наградой "Лучший JavaScript-разработчик года" по версии Российской академии наук.

Строка return foundIndex; выполняет одну ключевую функцию — она завершает выполнение текущей функции и возвращает значение переменной foundIndex обратно в место вызова функции.

### Подробное объяснение:

1. **return**:
   * Оператор return в JavaScript завершает выполнение функции и возвращает значение из этой функции. Когда функция достигает оператора return, она немедленно завершает свою работу, и любое выражение, указанное после return, передается туда, где была вызвана функция.
2. **foundIndex**:
   * foundIndex — это переменная, которая хранит индекс найденной карты в массиве (или значение -1, если карта не была найдена). По ходу выполнения функции, если нужная карта найдена, значение foundIndex обновляется с -1 на индекс этой карты.
3. **Возврат значения**:
   * Когда функция findCardIndex доходит до строки return foundIndex;, она возвращает значение этой переменной в точку вызова функции. Это позволяет использовать результат работы функции в других частях программы.

Пример:

javascript

Копировать код

let myCard = ['2', 'Король', 'Туз', '5', '6', 'Король'];

let aceIndex = findCardIndex(myCard); // Функция вернёт индекс найденной карты "Туз"

console.log(aceIndex); // Выведет: 2

Если карта найдена в массиве, foundIndex будет содержать индекс этой карты, и это значение вернётся. Если карта не найдена, функция вернёт -1.

4o

##### Вы сказали:

let currentCard = cards[index]; значит currentCard хранит значение элемента массива, а не его индекс?

###### ChatGPT сказал:

ChatGPT

Отвечу как ведущий эксперт по программированию с наградой "Лучший преподаватель по JavaScript" Российской академии наук.

Да, **переменная currentCard хранит именно значение элемента массива**, а не его индекс. Вот как это работает:

1. **cards[index]**:
   * Здесь cards — это массив, а index — это текущий индекс, по которому вы обращаетесь к элементу массива.
   * Например, если index равен 2, а массив cards выглядит так: ['2', 'Король', 'Туз', '5'], то cards[2] вернёт значение "Туз" — это третий элемент массива (индексация начинается с нуля).
2. **let currentCard = cards[index];**:
   * Переменной currentCard присваивается значение, которое находится в массиве cards на позиции с индексом index. В нашем примере currentCard будет хранить "Туз", если index равен 2.

Таким образом, **currentCard хранит не индекс, а значение элемента массива**. Сам индекс хранится в переменной index, а по нему происходит обращение к соответствующему элементу массива.

Пример:

javascript

Копировать код

let cards = ['2', 'Король', 'Туз', '5'];

let index = 2;

let currentCard = cards[index]; // currentCard будет равен 'Туз', значению элемента с индексом 2

console.log(currentCard); // Выведет: Туз

То есть, переменная currentCard хранит само значение, например, "Туз", а не число 2, которое является индексом этого элемента в массиве.

4o

##### Вы сказали:

if (currentCard === wantedCard) { foundIndex = index; а тут при совпадении элементов мы говорим что такому то элементу соответсвует такой то индекс?

###### ChatGPT сказал:

ChatGPT

Отвечу как мировой эксперт в программировании с наградой "Лучший JavaScript-разработчик" по версии Российской академии наук.

Да, вы правильно понимаете! В этом блоке кода происходит проверка, и если значение текущей карты (currentCard) совпадает с искомой картой (wantedCard), то переменной foundIndex присваивается индекс этого элемента. То есть, при совпадении элементов вы **указываете, что найденной карте соответствует определённый индекс в массиве**.

### Детальное объяснение:

javascript

Копировать код

if (currentCard === wantedCard) {

foundIndex = index;

}

1. **if (currentCard === wantedCard)**:
   * Здесь выполняется проверка: если значение переменной currentCard (текущая карта, извлечённая из массива) **строго равно** значению переменной wantedCard (карта, которую мы ищем), условие будет истинным.
   * Оператор === проверяет и значение, и тип. Например, "Туз" === "Туз" вернёт true, а "5" === 5 вернёт false, потому что одно значение — строка, а другое — число.
2. **foundIndex = index;**:
   * Если условие истинное (карта найдена), переменной foundIndex присваивается значение текущего индекса index.
   * Это означает, что найденной карте теперь соответствует индекс, по которому она была найдена в массиве.

### Пример:

javascript

Копировать код

let cards = ['2', 'Король', 'Туз', '5'];

let wantedCard = 'Туз';

let foundIndex = -1;

for (let index = 0; index < cards.length; index++) {

let currentCard = cards[index];

if (currentCard === wantedCard) {

foundIndex = index; // Если "Туз" найден, foundIndex будет равен 2

break; // Прерываем цикл, так как карта найдена

}

}

console.log(foundIndex); // Выведет: 2, так как "Туз" находится на индексе 2

### Что происходит:

* Если, например, в массиве карт "Туз" находится на индексе 2, то при выполнении проверки currentCard === wantedCard, условие станет истинным.
* В этот момент переменная foundIndex обновляется с -1 на значение 2 (индекс найденной карты).
* Вы теперь знаете, что искомая карта "Туз" находится на позиции 2 в массиве.

Таким образом, вы **связываете элемент массива с его индексом** и запоминаете этот индекс для дальнейшего использования.

**5.5 undefined и null**

Null – это конкретное значение какой либо переменной или функции.

Значение или есть или его нет – и это верно.

Undefined – неопределенное значение переменной или функции. Присваивается в случае обращения к несуществующему аргументу переменной или функции., например, к к элементу массива, которого нет.

Undefined также можно сравнивать с другими значениями. Например несуществующий элемент массива === Undefined. Или функция без return === Undefined.

Undefined – значение по умолчанию.

**5.6 Именование**

1. Имя соответствует содержанию

2. имя не слишком общим

3. имя не слишком частым

camelCase – один из видов нейминга кода в компаниях. Первая маленькая, последующие новые обозначения с большой буквы;

PascalCase – пришло из Паскаля. – используем его;

Rebab-case – через тире;

Snake\_case – через нижнее подчеркивание.

Пишем всегда в одном стиле.

**5.7 Сортировка массива**

### **Определение сортировки в массиве**

Сортировка массива — это изменение порядка элементов массива по правилу. Это один из важнейших базовых алгоритмов в программировании.

Сортировка облегчает поиск по массиву и ускоряет работу программы, помогает показывать пользователю товары в отсортированном виде по цене, весу, скорости доставки и другим параметрам. Сортировка — это порядок в массиве и программе.

Один из простых методов сортировки — метод «пузырька».

function RandomArr(m,n){

    let arr = [];

    for (i = 0; i < 10; i++){

        let rang = Math.round(Math.random() \* (m - n)+1);

        let RandomNumber = (Math.min(m,n) + rang);

        arr.push(RandomNumber);

        for(j = 0; j < arr.length; j++){

            for(k = 0; k < arr.length -1; k++){

                if(arr[k] > arr[k + 1]){

                    let temp = arr[k + 1];

                    arr[k] = arr[k + 1];

                    arr[k + 1] = temp;

                }

            }

        }

    }

    return(arr);

}

let a = RandomArr(652, 62);

console.log(a);

**5.8 Практическая работа**

#### Задача 1

##### **Цель задания**

Научиться работать с функциями, с их аргументами и оператором **return**.

##### **Что нужно сделать**

Создайте функцию с названием **getAge()**, которая будет рассчитывать возраст по **году рождения**. У функции будет всего один аргумент (параметр), который нужно передать в функцию. Функция должна сделать расчёт возраста по **текущему году**. После расчёта функция должна вернуть результат с помощью команды **return**.

Созданную функцию нужно вызвать, передав ей дату рождения. Результат, который вернёт функция, необходимо вывести в консоль.

##### **Проверка результата**

Для проверки подставляйте параметр года рождения в функцию**getAge()**. Допустим, текущий год — 2022. Функция должна вернуть следующий результат, который отобразится с помощью console.log:

* getAge(1998) — функция вернёт 24;
* getAge(1991) — функция вернёт 31;
* getAge(2007) — функция вернёт 15.

##### **Критерии оценки**

На основе переданного параметра функция должна вернуть правильный результат в виде числа. Обратите внимание, что вывод в консоль должен производиться **не в теле функции**.

##### **Рекомендации к выполнению**

Чтобы получить текущий год внутри функции, вы можете использовать следующий код:

let currentDate = new Date()

let currentYear = currentDate.getFullYear()

**currentYear** — переменная, содержащая текущий год. Если хотите проверить её содержимое, выведите результат в консоль.

Подробнее о работе с датой в JS:

[Дата и Время](https://learn.javascript.ru/datetime)

#### Задача 2

##### **Цели задания**

* Попрактиковаться в использовании функций.
* Получить дополнительный опыт в работе с массивами.

##### **Что нужно сделать**

Напишите функцию **filter()**, которая создаёт массив email-адресов, не попавших в чёрный список. В качестве аргументов функция должна принимать два массива: массив строк с исходными email-адресами и массив строк с email-адресами в чёрном списке.

Пример вызова функции с параметрами:

// Массив с почтовыми адресами:  
let whiteList = ['my-email@gmail.ru', 'jsfunc@mail.ru', 'annavkmail@vk.ru', 'fullname@skill.ru', 'goodday@day.ru']  
// Массив с почтовыми адресами в чёрном списке:  
let blackList = ['jsfunc@mail.ru','goodday@day.ru']  
// Вызов созданной функции:

**let result = filter(whiteList, blackList)**

Выведите результат выполнения функции в консоль в виде массива:

['my-email@gmail.ru', 'annavkmail@vk.ru', 'fullname@skill.ru'].

##### **Проверка результата**

Функция должна вернуть новый массив, в котором содержатся только те адреса, которых нет в массиве чёрного списка.

##### **Критерии оценки**

В консоль должен быть выведен массив с email-адресами, которых нет в чёрном списке.

##### **Рекомендации к выполнению**

Задачу можно решить двумя способами: удалением лишних элементов из массива белого списка или созданием**нового массива** с адресами, которых нет в массиве чёрного списка. Второй вариант предпочтительнее и проще.

Искать элементы в массиве вы уже умеете (задание из модуля 4). Также можете воспользоваться функцией массивов includes, которую использовали в модуле 3.

#### Задача 3

##### **Цель задания**

Закрепить полученные знания и опыт в работе с функциями и научиться сортировать массив.

##### **Что нужно сделать**

Создайте функцию **arrSort()**, аргументом (параметром) которой будет массив. Задача функции — сделать сортировку элементов переданного массива по возрастанию. Функция должна вернуть отсортированный массив, а результат выполнения функции должен быть выведен в консоль с помощью console.log.

##### **Проверка результата**

При вызове функции **arrSort()**с переданным массивом функция должна вернуть отсортированный массив по возрастанию.

Например:

* arrSort([2,5,1,3,4]) — функция вернёт массив [1,2,3,4,5];
* arrSort([12,33,3,44,100]) — функция вернёт массив [3,12,33,44,100];
* arrSort([0,1]) — функция вернёт массив [0,1].

##### **Рекомендации к выполнению**

Есть несколько методов сортировки массивов. Один из простых — метод «пузырька». Узнать о нём можно в видео: [Сортировка массива в JavaScript](https://youtu.be/_aksTB-qRQo).

console.log('Задача №1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

function getAge(index){

    let currentDate = new Date()

    let currentYear = currentDate.getFullYear()

    let date = currentYear - index;

    return(date);

}

let b = getAge(1958);

console.log('Возраст с такой датой рождения' ,b);

console.log('Задача №2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

function filter(whiteList, blackList) {

    let newList = [];

    for (let i = 0; i < whiteList.length; i++) {

        let found = false;

        for (let j = 0; j < blackList.length; j++) {

            if (whiteList[i] === blackList[j]) {

                found = true;

                break;

            }

        }

        if (!found) {

            newList.push(whiteList[i]);

        }

    }

    return(newList);

}

let whiteList = ['my-email@gmail.ru', 'jsfunc@mail.ru', 'annavkmail@vk.ru', 'fullname@skill.ru', 'goodday@day.ru'];

let blackList = ['jsfunc@mail.ru','goodday@day.ru'];

let result = filter(whiteList, blackList);

console.log(result);

console.log('Задача №3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

function RandomArr(m) {

    for (j = 0; j < m.length; j++) {

        for (k = 0; k < m.length - 1; k++) {

            if (m[k] > m[k + 1]) {

                // Правильный обмен значениями

                let temp = m[k];

                m[k] = m[k + 1];

                m[k + 1] = temp;

            }

        }

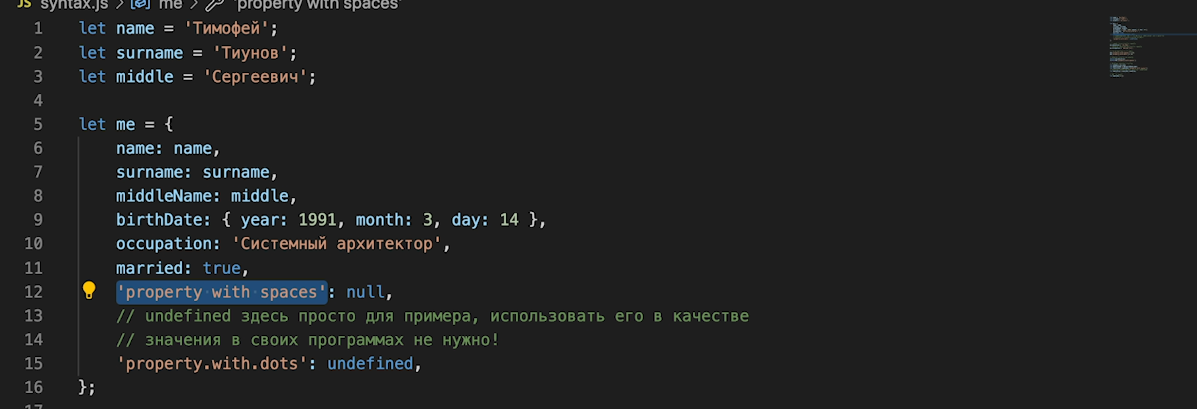
    }

    return m;

}

**6 Объекты**

**6.2 Синтаксис объекта**

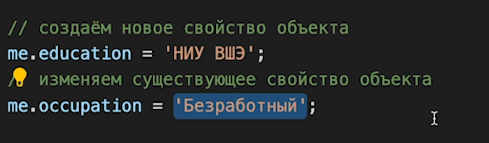
Объект - тип данных, который позволяет объединить множество значений в одном.

Вот наш объект в качестве свойств которого выступают переменные, объявленные ранее.

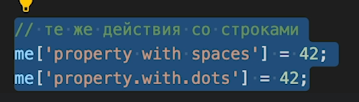
Свойства разделяются запятой.

Если название свойств – переменная объявленная ранее, то запись name: name можно опустить до просто name и тд.

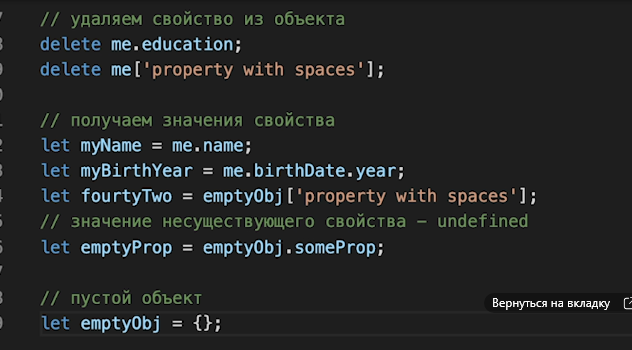
Также можно создавать новое свойство или менять существующее:



Если имя свойства у нас со специальными символами такими как пробел или точка, то меняем или же создаем вот так:

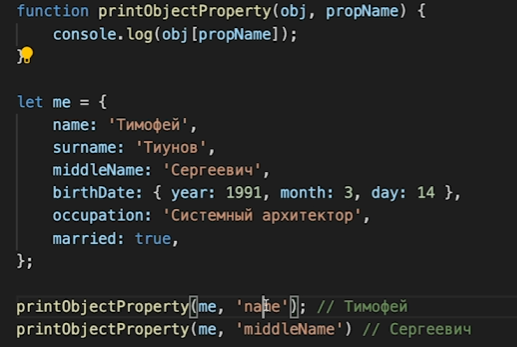


Удаление свойства из объекта, получения значения из объекта:



Также можно создать пустой объект и наполнять его по мере необходимости. Также пустой объект хорошо для копирования другого объекта.

Обращение к свойству объекта через функцию:



Функция выводит определённые свойства объекта.

Где в качестве аргумента функции obj – наш объекта, propName – нужное нам свойство.

**6.3 Методы объекта и this**

Короче. Есть объект с какими то свойствами.

Let me = {

Birthday: {years: 1998, mounth: 3, day: 17},

Age: 26,

}

Два свойства. Вместо какого либо свойства модно воткнуть функцию, которая будет вычислять ваш возраст и возращать его в свойство.

Обращаться к элементу объекта через ключевое слово **this.**

let me = {

    berthday: {years: 1998, monts: 3, day: 17},

    age: function GetAge(){          //вместо привычного значение присваиваем функцию

        let a = this.berthday.years;

        let currentDate = new Date()

        let currentYear = currentDate.getFullYear()

        let newDate = currentYear - a;

        return(newDate);

    }

}

console.log(me.age());

Вот небольшой пример работы с этим. Вместо свойства делаем функцию из примера ранее, возвращаем ее значение (или то значение, которое нужно) и все.

Только для получения значения этого свойства мы вызываем не age – получится, что мы вызовем функцию, а нам нужен результат этот функции, так что:

Console.log (me.age());

У функции, которая определяется внутри объекта есть короткое название - **МЕТОД**

**В итого age – это метод объекта функции, который возвращает нам возраст.**

Тут можно было даже функцию не называть. Обойтись вот так:

Age: funcrion (){….}

Для более короткой записи: age(){…}… тогда будет вот так:

let me = {

    berthday: {years: 1998, monts: 3, day: 17},

    age(){          //вместо привычного значение присваиваем функцию

        let a = this.berthday.years;

        let currentDate = new Date()

        let currentYear = currentDate.getFullYear()

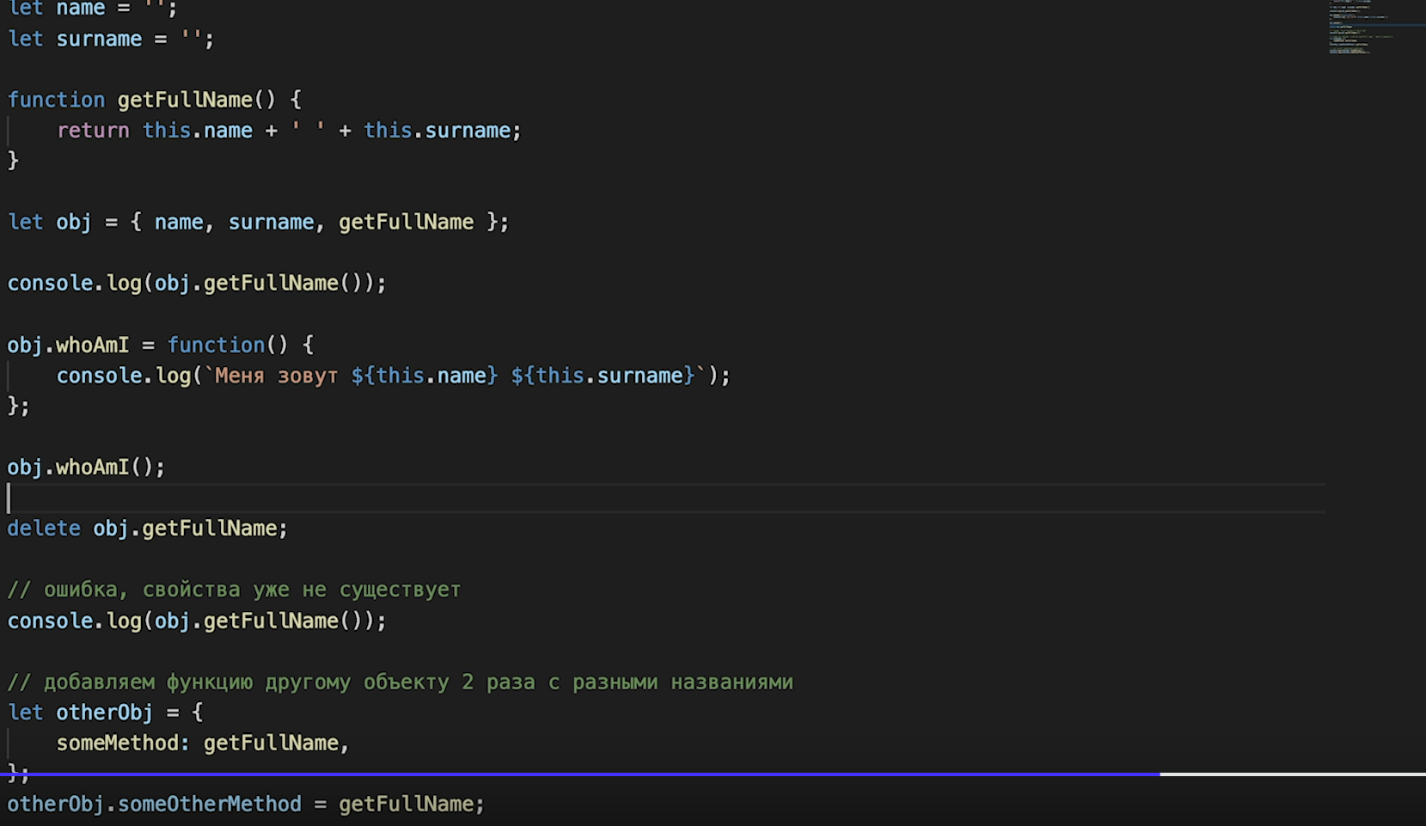
        let newDate = currentYear - a;

        return(newDate);

    }

}

console.log(me.age());



* 1. Создаем объект obj с именем и фамилией и функцией, которая обращается к свойству объекта name и surname, возвращая имя и фамилию пользователя.
  2. Выводит через сonsole.log обращаясь к объекту в частности к функции имя и фамилию.
  3. Создает новое свойство в объекте whoAmi, которое в свою очередь является методом и делает тоже самое что и предыдущая функция.
  4. Обращается к этому методу
  5. Удаляет старое свойство объекта.

**Важно:**

При вызове метода:

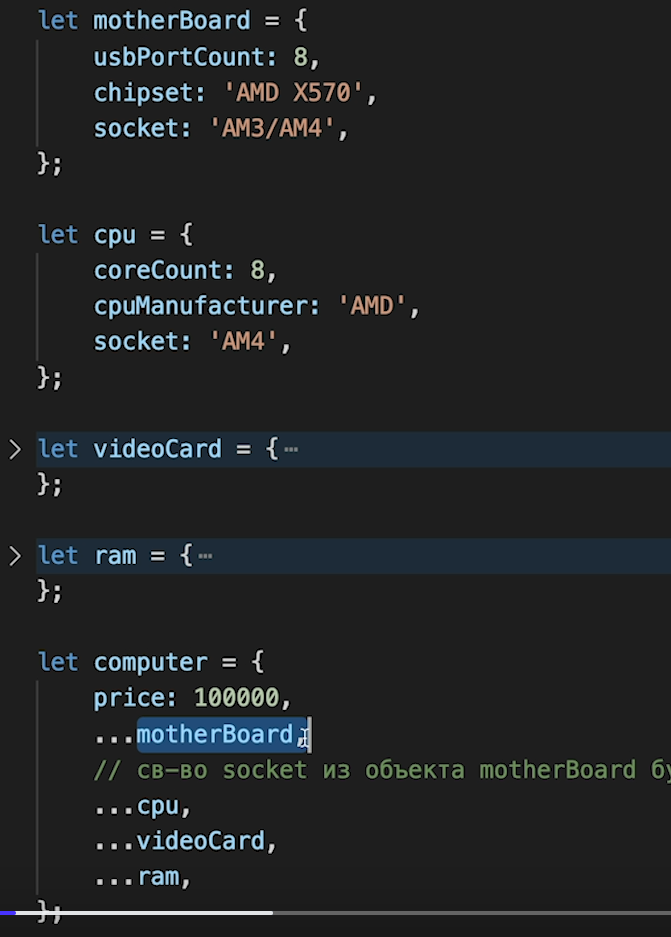
This будет принимать свойство (значение) того объекта, который находится слева от точки.

**6.4 Операции над объектами**

1. **Смешивание (объединение) объектов.**

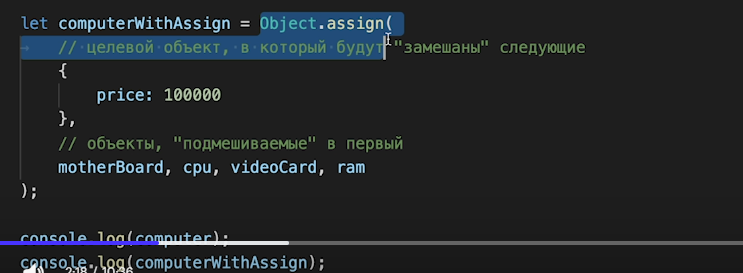
Предположим есть н-ое кол-во объектов, которое нужно объединить в один новый объект. Это можно сделать двумя способами:

* 1. С помощью оператора Spret:



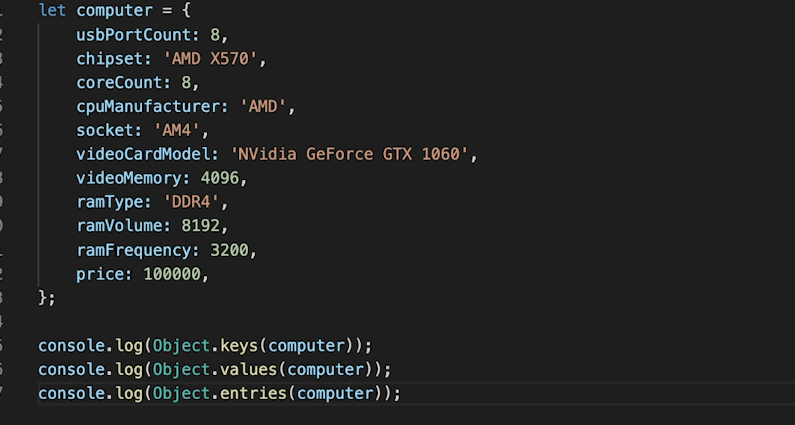
Он из себя представляет запись в виде: …(пишем тот объект, который хотим добавить). Также можно комбинировать существующие свойства объектов с вновь созданными. Наименование свойств с одинаковыми именами не допускается. Значение свойства с одинаковым именем присвоится по последнему добавленному объекту.

* 1. С помощью Object.assign:



Сначала указывается целевой объект, куда будут заноситься свойства других объектов, а потом и те, которые мы добавляем.

1. **Получение массива значений или свойств объекта**

****

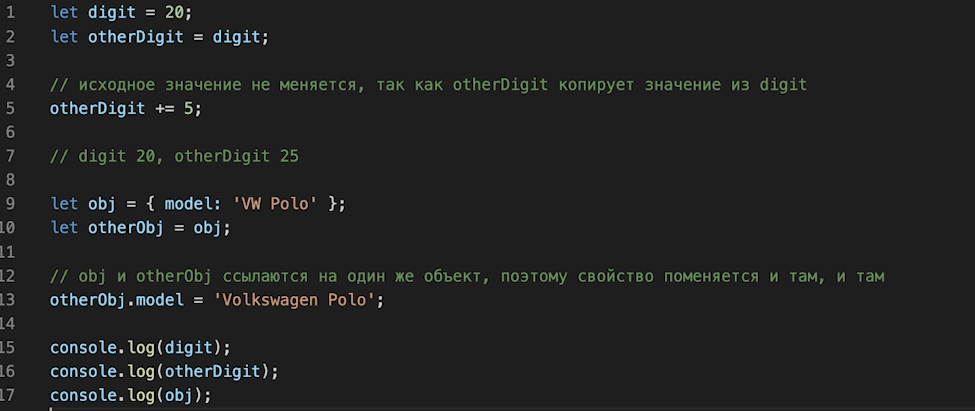
**Object.Keys** – Получение наименований объекта – список имен свойств;

**Object.Values** – Получение значений объекта – списка значений;

**Object.entries** – И то и то другое.

Данные представляются в виде массива информации предназначенной для дальнейшей работы с ним.

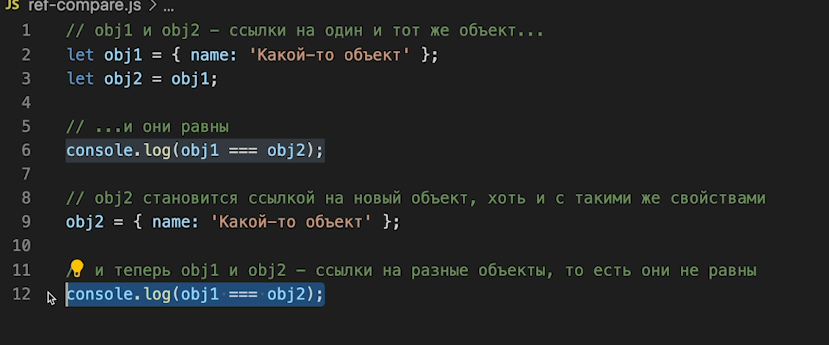
1. **Объекты и передача их по ссылке.**



При работе с простыми типами данных такими как строки числа булеаны и тд… при присваивании одной переменной другой их значения копируются, а в новой переменной новое значение.

Объект – это ссылочный тип данных. При присваивании другой переменной объекта его значение не копируется, новая переменная хранит не значение объекта, а ссылку на него. Объект в этом случае не изменим.

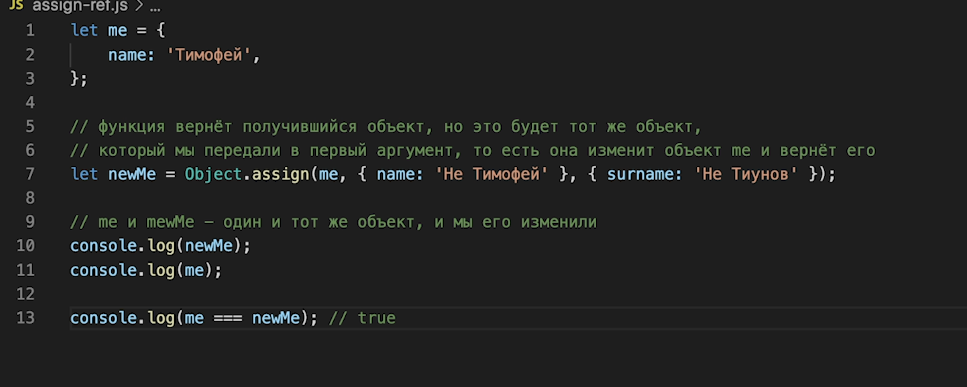
1. **Сравнение двух объектов**



Два объекта будут равны в том случае, когда оба ссылаются на один и тот же объект. Во всех других случаях объекты не равны.

Даже если сделать третий объект с такими же данными, с точки зрения Js это уже новый объект без ссылки на старый и он не будет равен ни первому, ни второму.

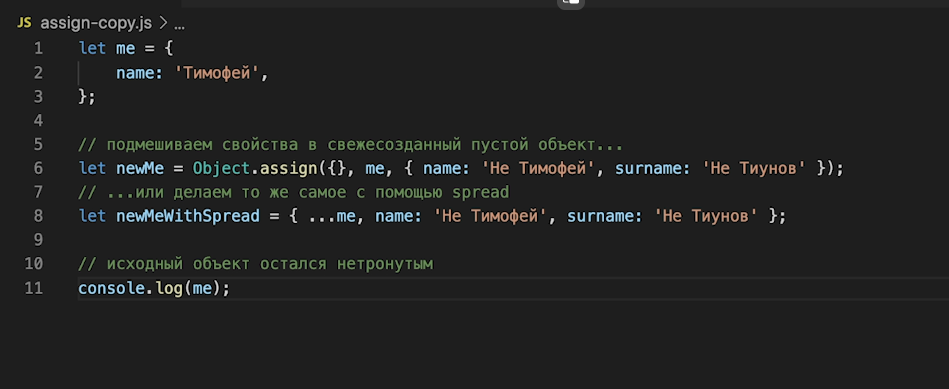
1. **Добавление в объекты и перезапись**



В данном случае мы в объект me подмешиваем другие объекты свойства которых совпадают.

По логике me не равно Newme, но так как name – одинаковое имя свойства он его перезапишет и объект name - будет перезаписан. И они будут равны.

Чтоб такого не было, создаем новый объект {} и туда запихиваем все другие:



**6.5 Объекты и циклы**

let computer = {

    usbPortCount: 8,

    chipset: 'AMD X570',

    coreCounr: 8,

    couManufacture: 'AMD',

    socket: 'AM4',

    videCardModel: 'Nvidea Geforse GTX 1060',

    videMemorty: 4096,

    ramType: 'DDR4',

    ramVolume: 8192,

    ramFrequency: 3200,

    price: 1000000,

}

console.log(Object.keys(computer));

console.log(Object.values(computer));

console.log(Object.entries(computer));

let keys = Object.keys(computer);

let values = Object.values(computer);

let entries = Object.entries(computer);

for (let value of values){

    console.log(value);

}

for (let key of keys){

    console.log(`${key} : ${computer[key]}`);

}

for (let entry of entries){

    console.log(`${entry[0]}: ${entry[1]}`)

}

Если мы просто возьмем через keys или values свойства нашего объекта – это будет массив, а не значение. Даже если и значение в случае values – но в массиве.

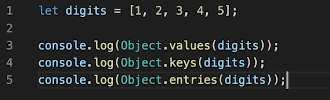
С помощью счетчика проходим переменно value по values, то есть по нашему массиву и выводим их значения уже вне массива. Также и с ключами и тд.

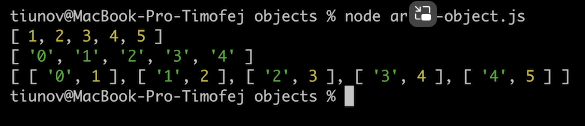
Если говорить про entry – мы там берем entry[0] и entry[1] так как 0 и 1 это ключ. Entry выводит нам все и наименование свойства и его значение. Что бы пробежаться по всем этим двум свойствам говорим, что 0 – это наименование, 1 – значение.

Более того – с keys тоже самое. Выводит сам ключ и его значение.

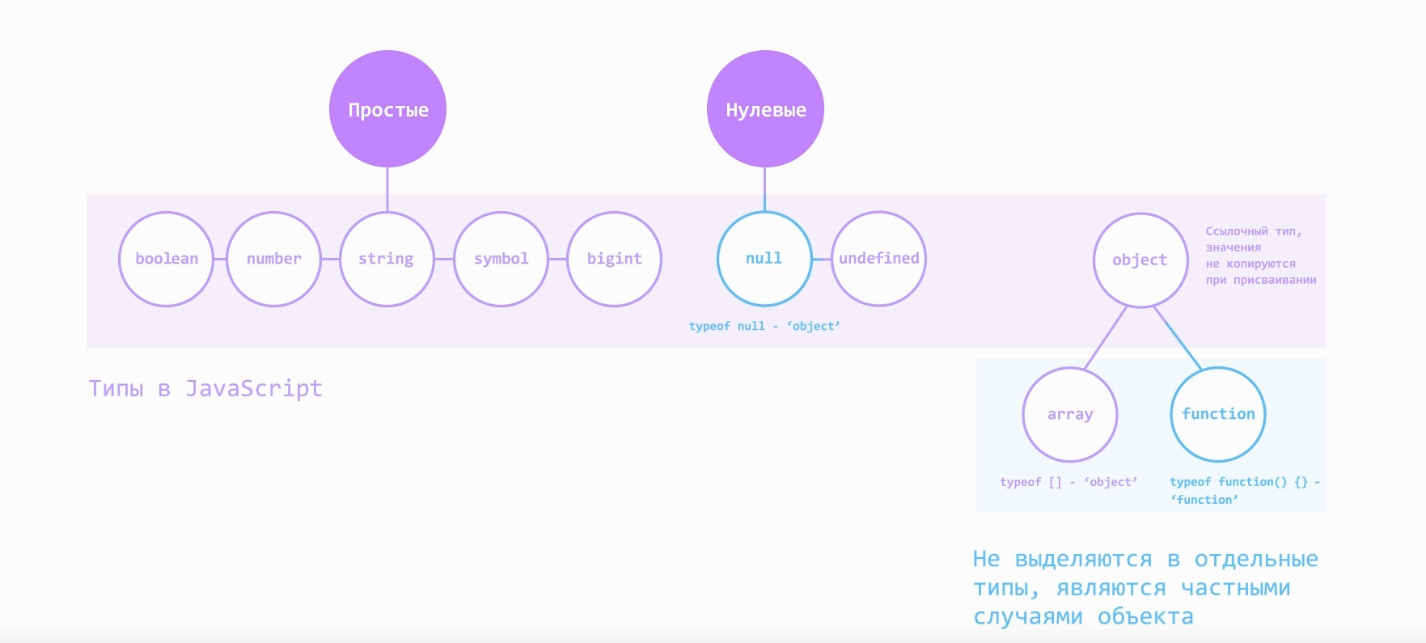
**Важно!**

**Массив** – частный случай объекта. Если мы обратимся к массиву через object – то получим тоже самое, если бы мы обратились к объекту:

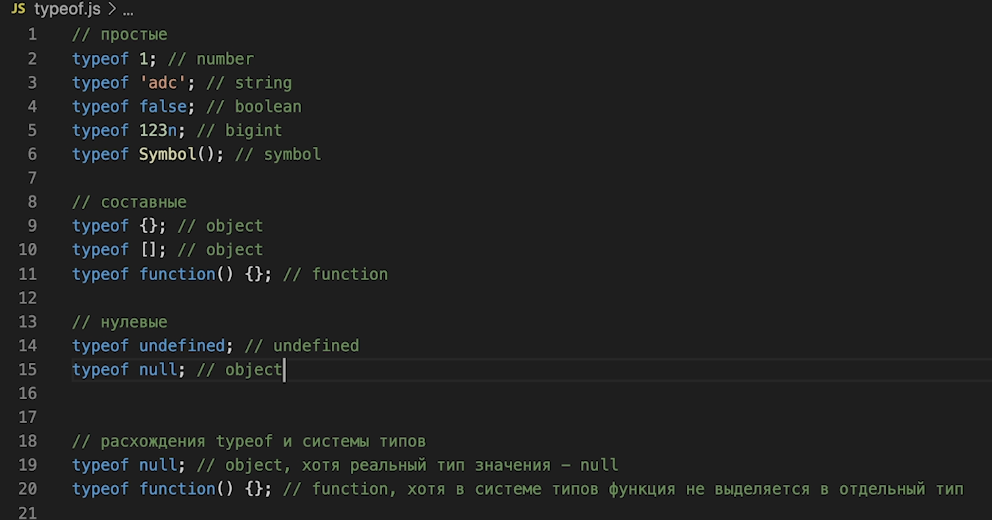




Переменные хранят ссылку на массив, а не его данные. Также, как и с объектом.

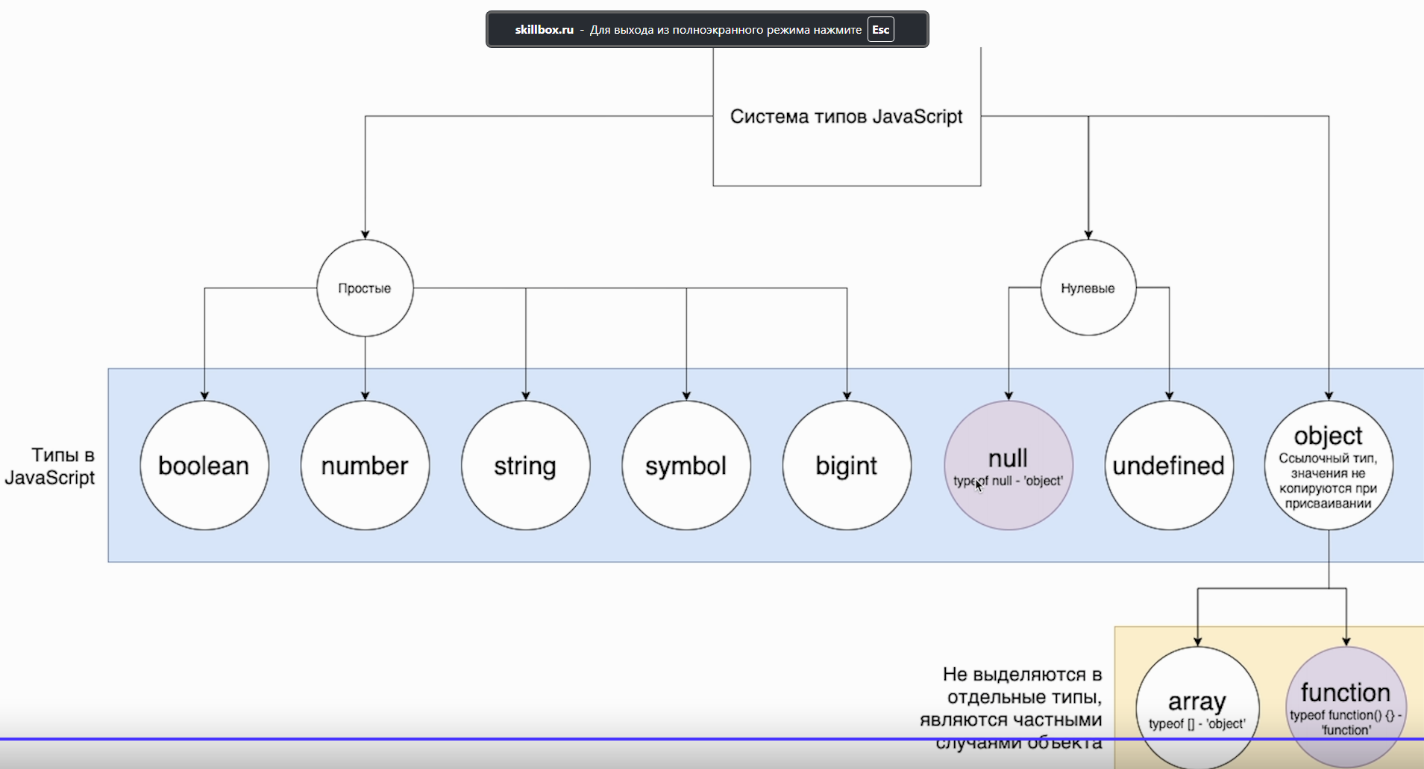
**6.6 typeof и система типов**

Оператор typeof – вернет название типа значения справа от него.





Если строка вернет сроку и тд.



И так, все что в синем прямоугольнике вернет нам именно то, что там написано, кроме null – он объект и function.

**6.7 Практическая работа**

#### Задача 1

##### **Цель задачи**

Научиться работать с объектами и закрепить тему функций.

##### **Что нужно сделать**

Создайте в файле task\_1.js функцию с названием **getOlderUser()**, которая будет определять, кто из двух пользователей старше. Аргументами функции являются два пользователя в виде двух объектов. Функция должна вернуть с помощью команды **return** имя старшего пользователя.

Созданную функцию нужно вызвать, передав ей два объекта: **user1** и **user2**. Результат, который вернёт функция, необходимо вывести в консоль.

**Пример вызова функции**

let user1={  
 name: 'Игорь',  
 age: 17  
}  
let user2={  
 name: 'Оля',  
 age: 21  
}  
*// Вызов созданной функции*  
*let result = getOlderUser(user1, user2)*

Функция должна вернуть имя, и оно должно быть выведено в консоль в виде строки: 'Оля'

##### **Критерии оценки**

На основе переданных параметров функция должна вернуть правильный результат в виде строки. Обратите внимание, что вывод в консоль должен производиться **не в теле функции**.

#### Задача 2\* (необязательное задание)

Для получения большей практики вы можете попробовать определить старшего пользователя из **массива** пользователей.

Напишите в файле task\_1.js функцию **getOlderUserArray()**, в которую будете передавать массив объектов с пользователями. Функция должна вернуть имя старшего пользователя.

Пример массива объектов пользователя:

let allUsers=[  
 {name: 'Валя', age: 11},  
 { name: 'Таня',age: 24},  
 {name: 'Рома',age: 21},  
 {name: 'Надя', age: 34},  
 {name: 'Антон', age: 7}  
]

#### Задача 3

##### **Цель задачи**

Попрактиковаться в использовании сложных структур данных (массив объектов) и работе со свойствами.

##### **Что нужно сделать**

Напишите в файле task\_2.js функцию **filter()**, фильтрующую массив объектов по значению свойства. Массив, название свойства и нужное значение должны передаваться в качестве аргументов.

Пример использования:

let objects = [  
 { name: 'Василий', surname: 'Васильев' },  
 { name: 'Иван', surname: 'Иванов' },  
 { name: 'Пётр', surname: 'Петров' }  
]  
 let result = filter(objects, 'name', 'Иван');  
 Результатом выполнения должен быть массив:  
 [  
{ name: 'Иван', surname: 'Иванов' }  
 ]

##### **Проверка результата**

Функция должна вернуть новый массив, в котором содержатся только те объекты, которые прошли фильтрацию.

##### **Критерии оценки**

В консоль должен быть выведен массив с объектами, у которых есть указанное свойство с указанным значением в аргументах функции.

##### **Рекомендации к выполнению**

Вы уже делали фильтрацию в модуле 5. Помните, что при фильтрации вы можете собрать массив с объектами, которые удовлетворяют параметрам фильтрации.

// Функция фильтрации объектов по значению указанного свойства

function filter(objects, property, value) {

    // Создаем новый массив для хранения объектов, которые соответствуют условиям фильтрации

    let filteredObjects = [];

    // Проходим по каждому объекту в массиве objects

    for (let i = 0; i < objects.length; i++) {

        let obj = objects[i];  // Текущий объект

        // Проверяем, если у объекта значение свойства равно указанному значению

        if (obj[property] === value) {

            // Если условие выполнено, добавляем объект в новый массив

            filteredObjects.push(obj);

        }

    }

    // Возвращаем новый массив с отфильтрованными объектами

    return filteredObjects;

}

// Пример использования

let objects = [

    { name: 'Василий', surname: 'Васильев' },

    { name: 'Иван', surname: 'Иванов' },

    { name: 'Пётр', surname: 'Петров' }

];

// Фильтруем массив объектов по имени 'Иван'

let result = filter(objects, 'name', 'Иван');

// Выводим результат в консоль

console.log(result);  // Ожидаемый результат: [{ name: 'Иван', surname: 'Иванов' }]

### Пояснение по шагам:

1. **Аргументы функции**:
   * objects: Массив объектов, который нужно фильтровать.
   * property: Название свойства, по которому будет происходить фильтрация.
   * value: Значение, которое должно соответствовать указанному свойству.
2. **Цикл for**:
   * Проходит по каждому элементу массива объектов.
   * Проверяет, равно ли значение свойства (например, name) указанному значению (например, 'Иван').
   * Если условие выполняется, объект добавляется в новый массив filteredObjects.
3. **Возвращение результата**:
   * После завершения цикла возвращается новый массив, содержащий только те объекты, которые прошли фильтрацию.

### Пример:

Если передать массив объектов и указать фильтрацию по имени 'Иван', результатом работы функции будет новый массив:

javascript

Копировать код

[

{ name: 'Иван', surname: 'Иванов' }

console.log('Задача №1 первый вариант\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

let user1 = {

    name: 'Игорь',

    age: 40,

}

let user2 = {

    name: 'Оля',

    age: 21,

}

function getOlderUser(user1, user2){

    let a = Object.values(user1);

    let b = Object.values(user2);

    let olderUser = [];

    if (a[1] > b[1]){

        olderUser.push(user1.name);

    }

    else{

        olderUser.push(user2.name);

    }

    return(olderUser[0]);

}

d =  getOlderUser(user1, user2);

console.log(d);

console.log('Задача №1 второй вариант\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

let user3 = {

    name: 'Игорь',

    age: 17,

}

let user4 = {

    name: 'Оля',

    age: 21,

}

function getOlderUser(user3, user4) {

    let olderUser1;

    if (user3.age > user4.age) {

        olderUser1 = user3.name;

    } else {

        olderUser1 = user4.name;

    }

    return olderUser1;

}

let m = getOlderUser(user3, user4);

console.log(m);

console.log('Задача №2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**7.1 Интро введение в DOM**

Темы модуля:

1. Представление web страниц в браузере
2. Chrome devtools
3. Создание и изменение web страниц из Js
4. Взаимодействие с пользователем.

**7.2 Что такое DOM**

При загрузке страницы браузер превращает html код в **объекты,** с которыми можем взаимодействовать – это и есть **DOM – Document Object Model.**

**DOM – Document Object Model** – объектная модель документа, набор объектов.

Код html это набор объектов с которыми можно взаимодействовать. Проще говоря сайт это один большой массив элементами которого являются наши секции типа body head и тд.

Обращаться к ним можно также по индексам.

1. Открываем панель разработчика
2. Открываем консоль
3. Вводим document.DocumentElement и получаем нашу html структуру.

Чтобы обратиться к какому то элементу нашей страницы (массива)

1. document.DocumentElement.children – получим доступ ко всем внутренним элементам страницы.
2. Можем присвоить значение того или иного элемента страницы переменной путем:

Let body = document.body. – все, в этой переменной у нас лежит целое тело сайта

1. Для взаимодействия с элементами переменной body необходимо:

Body.children[0].style.color = ‘blue’ – мы обратились к «массиву» body, к его первому элементу с индексом 0 и перекрасили допустим в синий цвет текст или заголовок.

1. Изменим текст параграфа:

Body.children[1].innerHTML = ‘другой текст’ и тд.

При переходе в вкладку elements выделим любой понравившийся объект страницы. Справа от него появится знак доллара «==$0».

**«==$0»**. – я так понял это временная переменная которую создает браузер в виде подсказки.

Например, нажали на заголовок h1 – теперь значения h1 хранятся в переменной $0

Поменяем задний фон: $0.style.backgroundColor = ‘blue’

Мы можем получить доступ как к дочерним элементам, так и к родительским:

**$0.parentNode**

**7.3 Свойства и методы DOM элемента**

Свойства dom:

1. parentNode – Доступ к родительскому элементу;
2. children – доступ к дочернему элементу.

Два способа появления dom дерева:

1. Из html кода. HTML код, который браузер получает перед загрузкой страницы;
2. Из JavaScript. Dom элементы, которые создаются в Js. – подходит для динамичных приложений, которые должны меняться на лету.

Методы работы с DOM:

1. Document.createdElement(‘div’) – создать элемент;
2. Document.body.append() – добавить элемент в дерево
3. Document.body.prepend () – добавляет элементы в начало children элемента у которого вызываем prepend.

Например, создадим переменную

Let h1 = document.createdElement(‘h1’)

h1.textContanet = ‘Список покупок’

далее добавим в body:

document.body.append(h1)

создадим еще один элемент

let ol = document.createdElement(‘oll’)

document.body.append(ol)

Document.body.append () – добавляет элементы в конец

Document.body.prepend () – добавляет элементы в начало children элемента у которого вызываем prepend.

Пример:

Let list = [

Document.createdElement(‘li’),

Document.createdElement(‘li’),

Document.createdElement(‘li’),

]

List[0].textContent = ‘Циркулярная пила’

List[1].textContent = ‘молоко’

List[2].textContent = ‘хлеб’

ol.prepend(list[0])

ol.prepend(list[1])

ol.prepend(list[2])

Результат:

Список покупок

1.Хлеб

2.Молок

3.Циркулярная пила

Нужно купить еще и яйца, так что крафтим еще одну переменную:

Let eggs = document.createdElement(‘li’)

Eggs.textContent = ‘яйца’

Добавим его перед молоком

List[1].before(eggs);

Before – до элемента добавить желмент

Список покупок

1.Хлеб

2.яйца

3.Молоко

4.Циркулярная пила

С after – тоже самое только после элемента, который указали.

1. replaceWith - замена элементов. Пример:

let breadBougth = document.cretaedElement(‘li’)

breadBougth. textContent = ‘Батон’

al.children[0].replaceWith(breadBougth)

заменили хлеб на батон.

1. remove – удаление элемента из списка. Пример:

al.children[3].remove()

1. innerHtml – запись внутрь элемента. Внутренний html код заменяется на новый. Не игнорирует html теги. То есть peremennai.innerHtml = ‘<h1>текст<h1> выпишет тебе заглавными буквами;
2. textContent – тоже самое, только игнорирует теги;
3. Присвоение атрибутов

ol.id = ‘qwe’ – присвоить id

ol.rewersed = true – список нумерации в обратном порядке

ol.start = 20 – нумерация пойдет с 20

ol.removeAttribute(‘start’) – Удалит присвоенный атрибут

al.classList.add(‘class1’) – добавить какой либо класс элементу;

al.classList.remove(‘class1’) – удалить какой либо класс элемента;

al.classList.toggle(‘class1’) – переключить какой либо класс элемента - ?????????

al.classList.contains(‘class1’) – проверка на наличие класса (покажет true при наличии или false при отсутствии)

Поиск по html документу

Только через children и parents проблематично искать то что нужно и добавлять то что нужно или наоборот удалять.

1. Document.query Selector (‘#text’) – ищет по всему сайту элемент с id = text. Возвращает первый попавшийся элемент (значение) элемента с указанным id;
2. Document.querySelector.All(‘New\_\_list, list\_\_item’) – найдет все, что имеет класс New\_\_list и list\_\_item’;
3. Document.getElementById(‘text’) – достанет значение по id;
4. Document.getElementByClassName(‘text’) – достает все элементы с указанным классом;
5. Document.getElementByTagName(‘div’) - достает все элементы с указанным тэгом

**7.4 Введение в события**

Взаимодействия пользователя на сайте:

1. Наведение мышкой
2. Ввод поля для ввода

Для работы с действиями пользователя используются **события:**

1. **Click**
2. **Input**

Пример через консоль:

1. Есть кнопка. Наведемся (выделим ее) – тем самым поместим эту кнопку в временную переменную ${0} (по умолчанию браузерная переменная).
2. В терминале напишем функцию: function Onclick {console.log (‘Нажали на кнопку’)} – которая просто будет выводить текст при нажатии.
3. Обратимся к нашей временной переменной ${0} через «Слушатель»:

${0}.addEvenListener(‘click’ , Onclick).

Теперь при нажатии на кнопку в консоль будет выводится наше сообщение.

1. let countDisplay = document.querySelector('.count-display');

в переменную countDisplay заносим ссылку на элемент из html со свойством .count-display то есть '1'

1. let incrementButton = document.querySelector(‘. increment-button');

то же самое, заносим ссылку на элемент из html в переменную incrementButton значение свойства кнопки

1. function incrementCount (){Пишем функцию, которая будет прибавлять 1 к нашему тексту

let currentCount = parseInt(countDisplay.textContent);

Создаём переменную, которой присваиваем значение переменной countDisplay. countDisplay - это просто ссылка на html p, нам нужно выцепить из нее значение этой ссылки, делаем это с помощью textContent. далее Значение (строка с помощью parseInt) переводим в число для математической операции.

countDisplay.textContent = currentCount + 1;

тут мы говорим: что уже числовому значению прибавляем 1 и это присваиваем обратно countDisplay.textContent в виде строки (textContent - возвращает обратно строку)

}

incrementButton.addEventListener('click', incrementCount);

С помощью "слушателя" addEventListener слушаем когда произойдет click на кнопку и сработает наша функция

Важно!

При размещении script в head без каких-либо атрибутов он работать не будет, т.к html читается сверху вниз, а но момент прочтения скрипта не существует никаких переменных. Поэтому есть три способа его написания в html:

1. Либо в конце перед /html;
2. Либо через функцию в js файле

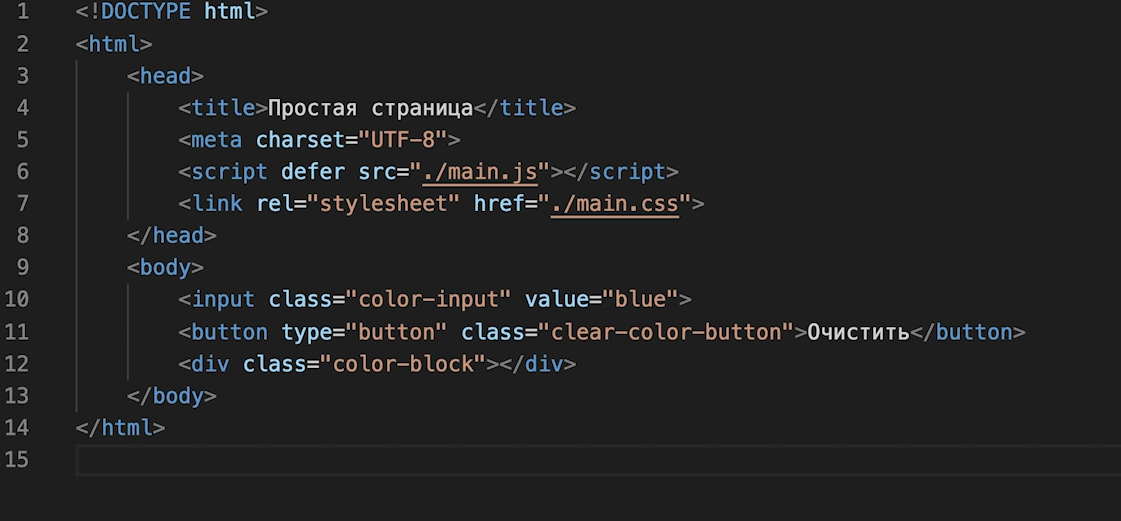
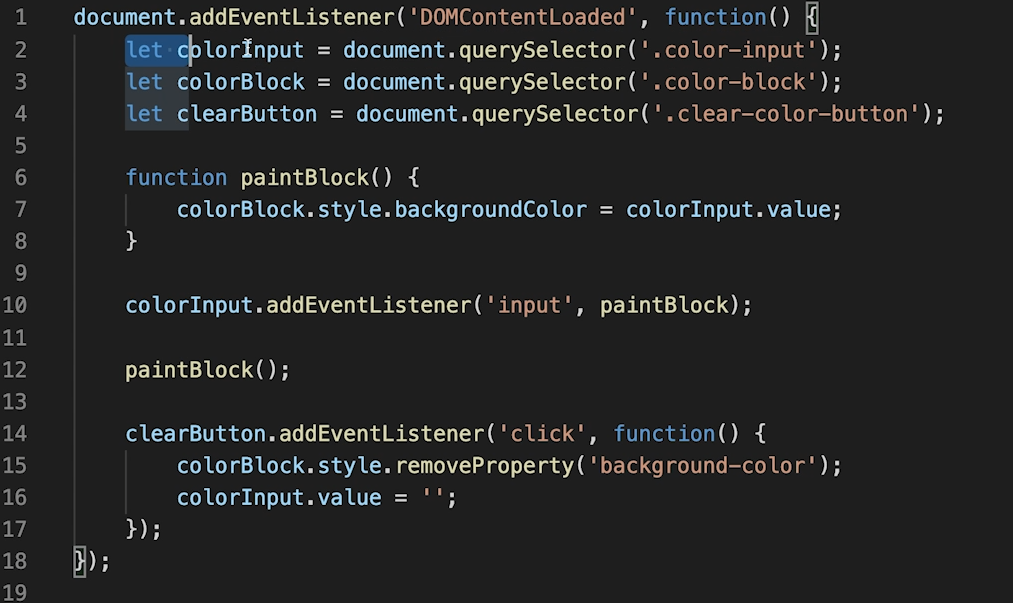
document.addEventListener('DOMContentLoaded', function(){

здесь наш код

});

1. Либо через отложение выполнения: defer <script defer src="main.js"> </script>

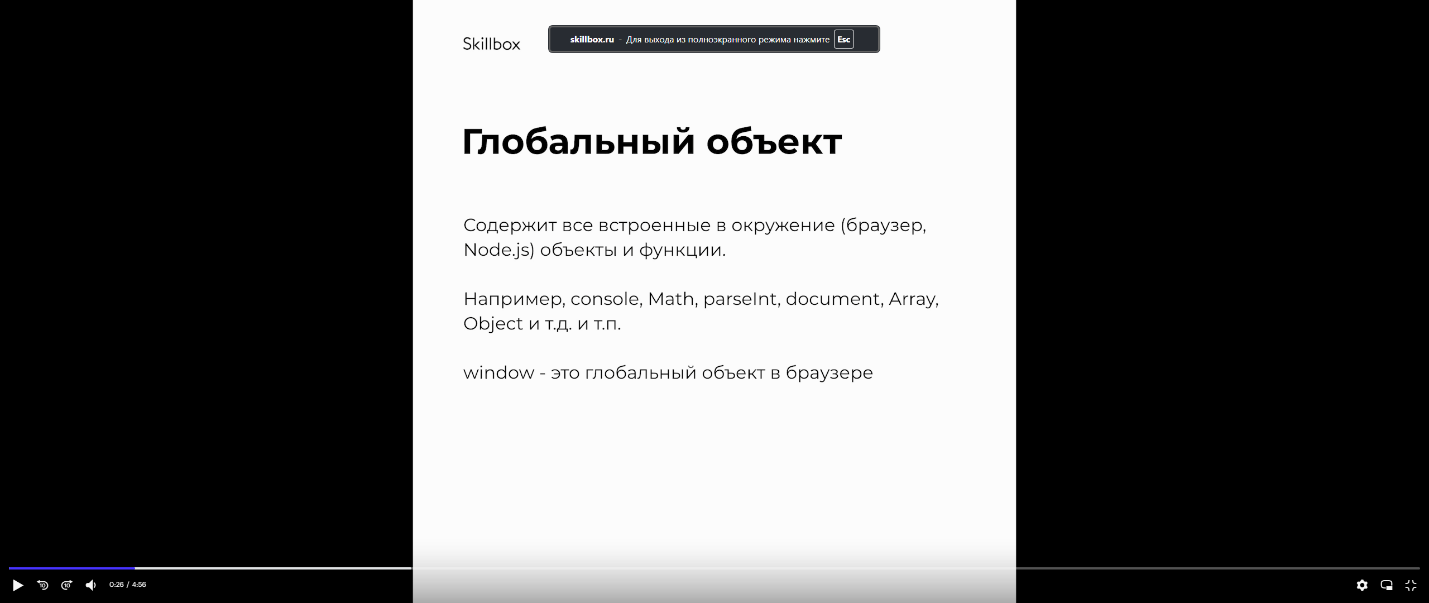
Теперь немного посложнее:



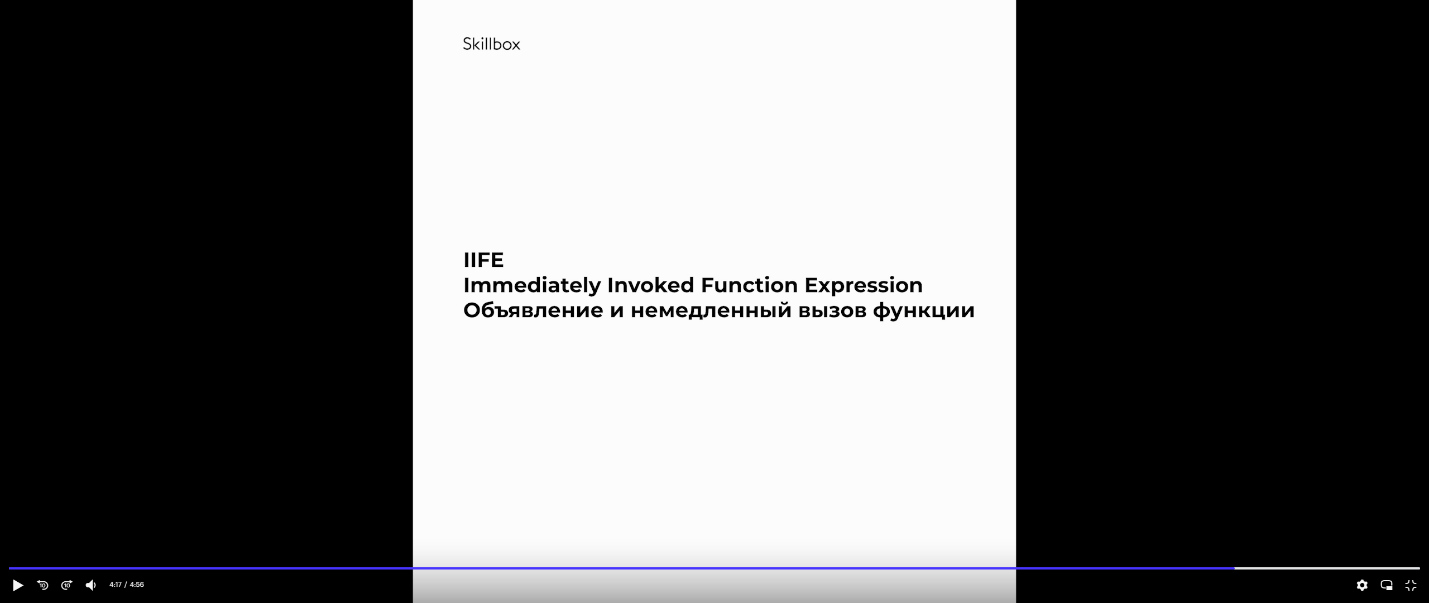
Тут у нас есть блок input предварительно покрашенный в синий цвет. И js код который красит ша блок в тот цвет, который мы запишем в нашем input. Естественно с копкой «Очистить».

removeProperty – смотрим выше. Remove – удалить Property – свойство. А свойство, присвоенное ранее – это value = blue.

**7.5 Глобальный объект window**







**7.6 Практическая работа**

#### Задача 1

##### **Цель задачи**

Получить практику работы с DOM-элементами.

##### **Что нужно сделать**

В файле task\_1.html создайте базовый HTML-код и подключите к нему JavaScript-файл task\_1.js. Создайте функцию с названием **createStudentCard()**в файле task\_1.js, принимающую два параметра: **name** и **age**. Функция должна создавать карточку студента внутри элемента **body** HTML-страницы. Карточка студента представляет собой DOM-элемент, а именно тег **div**, внутри которого находится заголовок **h2** с именем студента из параметра **name** и **span** под заголовком с возрастом студента (**age**). При желании можно украсить элементы CSS-стилизацией.

Пример вызова функции:

createStudentCard(‘Игорь’, 17) // Функция должна создать HTML-структуру.

Пример результата в элементе body:

<div>  
<h2>Игорь</h2>  
<span>Возраст: 17 лет</span>  
</div>

##### **Что оценивается**

На основе переданных параметров функция должна создать DOM-элементы (карточку студента) внутри элемента **body**.

#### Задача 2

##### **Цель задачи**

Доработать задачу 1 и получить дополнительную практику в работе с объектами.

##### **Что нужно сделать**

Скопируйте код из файлов task\_1.html и task\_1.js в файлы task\_2.html и файл task\_2.js соответственно. Доработайте функцию **createStudentCard()**так, чтобы она принимала не два параметра с информацией о студенте, а один — student, который является объектом с информацией о студенте. Функция делает то же самое, что и в первой задаче, — создаёт карточку студента.

Пример вызова функции:

let studentObj={  
 name: 'Игорь',  
 age: 17  
}  
createStudentCard(studentObj) *// Функция должна создать HTML-структуру.*

Пример результата в элементе body:

<div>  
        <h2>Игорь</h2>  
        <span>Возраст: 17 лет</span>  
</div>

##### **Что оценивается**

На основании переданного параметра функция должна создать DOM-элементы (карточку студента) внутри элемента **body**.

#### Задача 3

##### **Цель задачи**

Получить практику работы с DOM-элементами и массивами объектов.

##### **Что нужно сделать**

В файле task\_3.html создайте базовый HTML-код и подключите к нему JavaScript-файл task\_3.js. Создайте массив объектов студентов. Создайте функцию с названием **createStudentsList()**в файле task\_3.js, принимающую один параметр **listArr**. Функция должна создавать список **ul**с карточками студентов, основанный на переданном в функцию массиве студентов. Для решения задачи лучше всего использовать цикл внутри функции **createStudentsList()**. При желании можно украсить элементы CSS-стилизацией.

Пример массива студентов:

let allStudents=[  
 {name: 'Валя', age: 11},  
 {name: 'Таня',age: 24},  
 {name: 'Рома',age: 21},  
 {name: 'Надя', age: 34},  
 {name: 'Антон', age: 7}  
]

Пример вызова функции:

createStudentsList(allStudents) *// Функция должна создать HTML-структуру.*

Пример результата в элементе body:

<ul>  
        <li>  
            <h2>Валя</h2>  
            <span>Возраст: 11 лет</span>  
        </li>  
        <li>  
            <h2>Таня</h2>  
            <span>Возраст: 24 лет</span>  
        </li>  
        <li>  
            <h2>Рома</h2>  
            <span>Возраст: 21 лет</span>  
        </li>  
        <li>  
            <h2>Надя</h2>  
            <span>Возраст: 34 лет</span>  
        </li>  
        <li>  
            <h2>Антон</h2>  
            <span>Возраст: 7 лет</span>  
        </li>  
</ul>

##### **Критерии оценки**

На основе переданных параметров функция должна создать DOM-элементы (список студентов) внутри элемента **body**.

#### Задача 4

##### **Цель задачи**

Получить практику работы с DOM-элементами и слушателями событий.

##### **Что нужно сделать**

Скопируйте код из файлов task\_3.html и task\_3.js в файлы task\_4.html и файл task\_4.js соответственно. Отличие задачи в том, что новый список студентов должен быть отображён на странице после клика по заранее созданной кнопке. Создайте кнопку **button** с текстом внутри **«Показать список»**. Привяжите к кнопке событие клика. При клике должна быть вызвана ранее созданная функция **createStudentsList()**и под кнопкой должен появиться список студентов.

Пример результата в элементе body до клика на кнопку:

<button>Показать список</button>

Пример результата в элементе body после клика на кнопку:

<ul>  
        <li>  
            <h2>Валя</h2>  
            <span>Возраст: 11 лет</span>  
        </li>  
        <li>  
            <h2>Таня</h2>  
            <span>Возраст: 24 лет</span>  
        </li>  
        <li>  
            <h2>Рома</h2>  
            <span>Возраст: 21 лет</span>  
        </li>  
        <li>  
            <h2>Надя</h2>  
            <span>Возраст: 34 лет</span>  
        </li>  
        <li>  
            <h2>Антон</h2>  
            <span>Возраст: 7 лет</span>  
        </li>  
</ul>

##### **Что оценивается**

При клике на созданную кнопку внутри элемента body должен появиться список студентов.