**ARRAYS METHODS:**

let nums = [1, 2, 3, 4, 5]; - деклариране на масив nums

nums.indexOf(2) - Дали и на коя позиция има числото 2 в nums

* Returns 1 - Връща 1 , че има на позиция 1 (дава само първото намерено число)
* If not found returns -1 - Връща -1 (false) ако го няма числото в масива

nums.includes(5) - Дали има числото 5 в nums

* Returns true if fiund - true ако има
* Otherwise false - false ако няма

nums.push(6) - Добавяне на елемент(и) в края на масива, напр. nums.push(6, 7, 8);

* Adds an element(s) at the end
* nums = [1, 2, 3, 4, 5, 6];

nums.unshift(6) - Добавяне на елемент(и) в началото на масива

* adds an element(s) at the start
* nums = [6, 1, 2, 3, 4, 5, 6];

nums.pop() - Премахва последния елемент и връща стойността му

* removes from the end and returns the element
* let last = nums.pop(); last = 6; nums = [6, 1, 2, 3, 4, 5];

nums.shift() - Премахва първия елемент и връща стойността му

* removes from the start and returns the element
* let first = nums.shift(); first = 6; nums = [1, 2, 3, 4, 5];

nums.slice(startIndex,endIndex) – Връща масив с елементи от startIndex до endIndex (не включително)

* nums.slice(startIndex) – само със старт индекс: връща от индекса до края на масива.Същото го има и при splice.
* returns a new array
* let newNums = nums.slice(1,3); newNums = [2, 3];
* let numsCopy = nums.slice(0); numsCopy = [1, 2, 3, 4, 5]; - Начин за създаване на копие на масив.

nums.splice(startIndex, count) - Променя масива и връща изрязка от startIndex с размер count

* modifies the array and returns the sliced part
* nums.splice(1, 4); nums = [2, 3, 4, 5];

nums.splice(1, 0, 8) - На позиция 1, отрязва нищо 0, добавя число(а) 8, 9, 10

* insert 2 at index 1
* nums = [2, 8, 3, 4, 5];

nums.forEach((num, index) => console.log(`${num}, ${index}); - обхожда всички елементи, може да покаже и индекс

nums.map((n) => n \* 2) - Умножава всички елементи от масива по 2 и връща нов масив.

* doubles all numbers and returns a new array (conversion function) - Конвертиране на всички елементи.
* let doubledNums = nums.map((n) => n \* 2); doubledNums = [4, 16, 6, 8, 10];
* let numsAsStr = [‘1’, ‘2’, ‘3’]; let convertedNums = numsAsStr.map(Number); convertedNums = [1, 2, 3];
* let powerNums = convertedNums.map((n) => n \*\* 3); powerNums = [1, 8, 27]; - Повдигане на степен става с оператор (\*\*)
* powerNums = powerNums.map((n) => n + ‘’); powerNums = [‘1’, ‘8’, ‘27’]; - От числа към стринг; може и с .toString();

nums.filter((n) => n % 2 === 0) - Намира всички четни числа и връща нов масив (премахва всички нечети елементи)

* аргумента е булев(true, false). Retrieves only all even numbers and returns a new array
* let evenNums = nums.filter((n) => n % 2 === 0); evenNums = [2, 8, 4];

nums.sort() - Сортиране по ASCII (по азбучен ред);

nums.sort((a, b) => a – b) - Сортира във възходящ ред; nums.sort((a, b) => b – a) - сортира в низходящ ред (Големи-малки)

let array = [[1, 2, 3,], [[4, 5, 6], [7, 8, 9], [[10, 11, 12], [13, 14, 15], [16, 17, 18]]];

console.log(array[1][0][2]); // 6 адресиране на елемент в многомерен масив – всеки следващ индекс бърка по-навътре.

Array.isArray(nums); // true проверка дали е масив

. **\** - ескейп символ в JS

**For-of (forof) e за масиви, обхожда стойностите (values)**

**For-in (forin) обхожда индексите 0,1,2.3…. сойностите се достъпват с object[i]**

**ОБЕКТИ И КЛАСОВЕ**

Let primerenObject = {  
 primeren: true, // property(или key): стойности(параметри),

fileExtention: “PDF”,  
};

Методи се записват така: Object.keys(imeNaObekta)

**Class – шаблон, Objects – могат да се извадят по шаблона;**

Class **ImeSGlavnaBukva** { **- дефиниране на клас**

nqkakvoProperty = ‘neshto’; - обикновено няма стойности при дефинирането

nqkakvoDrugoPrperty = true;

}

Let ImeNqkakvo = new ImeSGlavnaBukva(); - нов обект, по шаблона на класа

ImeNqkakvo.nqkakvoProperty = ‘razlichno neshto’; - неговите си нови стойности на шаблонните пропъртита

ImeNqkakvo[nqkakvoProperty] = ‘razlichno neshto’;

class mity { - клас със конструктор

constructor(name, area, population, country, postCode) {

this.name = name;

this.area = area;

}

}

let city = new mity(name, area);

Object.entries(city).forEach(tuple => console.log(`${tuple[0]} -> ${tuple[1]}`));

**ОСНОВНИ**

* Objects hold **key-value pairs**
  + Access key and value **by index** in loops
  + Access value with **[**'key name'**]**
  + Access value with **obj.key**
* Use Object **Methods** such as:
  + **Object.keys - масив/Array от ключове**
  + **Object.values - масив/Array от стойности**
  + **Object.entries - масив/Array от тюпъли (двумерен масив)**
* **Parse** and **stringify** objects in **JSON**

a = [ **// Намиране на позицията на елемент на обект вътре в масив**

{prop1:"abc",prop2:"qwe"},

{prop1:"bnmb",prop2:"yutu"},

{prop1:"zxvz",prop2:"qwrq"}];

index = a.findIndex(x => x.prop2 ==="yutu");

console.log(index);

**// ПРИНТИРАНЕ НА ОБЕКТ**

Object.entries(products).forEach((product) => console.log(`${product[0]} -> ${product[1]}`));

Object**.keys**(obj)**.length** **// Намиране на дължината на обект**

Превръща обекта в масив по ключове и намира дължината му.   
Ако ползваме стойности или entries и филтрираме по **undefind** ще получим ентрита **без стойности undefind:**

Object.values ( {a: undefined} ) .filter ( value => value ).length // .filter ( value => value ) връща true/false дали има сойност на key-a

**СОРТИРАНЕ НА ОБЕКТИ**

**// СОРТИРАНЕ НА ОБЕКТИ (ПО СТОЙНОСТ) С ЕДНАКВИ КЛЮЧОВЕ**

**function sortObject() {**

**const data = [['over 18', { age: 24 }], ['under 18', { age: 12 }]];**

**let sorted = [...new Map(data)] // [...спредва се] до масив от тюпъли**

**.map(tuple => tuple[1]) // взимаме вторите им елементи[1] - {обектите}**

**.sort((a,b) => b.age - a.age); // сортираме по ключ .age**

**console.log(sorted); // принтираме масив от сортирани обекти**

**}**

**sortObject()**

**// Сортиране на два обекта (по стойност) с различни ключове**

**function sortObj() {**

**const data = [['over 18', { john: 24 }], ['under 18', { simon: 12 }]];**

**let sorted = [...new Map(data)] // [...спредва се] до масив от тюпъли**

**.map(tuple => tuple[1]) // взимаме вторите им елементи[1] - {обектите}**

**.sort((a,b) => b.john - a.simon); // сортираме по ключ .age**

**console.log(sorted); // принтира масив от сортирани обекти**

**}**

**sortObj()**

**// СОРТИРАНЕ НА МАСИВ ОТ ОБЕКТИ (по стойност) с еднакви ключове**

function sortObject(mine) {

let sorted = mine //1. взимам масива от обекти

.sort((a,b) => b.age - a.age); //2. сортирам по ключ .age

console.log(sorted); //3. принтирам масив от сортирани обекти

}

sortObject(arrayObj)

**// СОРТИРАНЕ НА МАСИВ ОТ ОБЕКТИ ПО ДВА КРИТЕРИЯ 05. Sort an Array by 2 Criteria.js**

function sortObject(mine) { //1. mine е масивът, направен с class constructor

let sorted = mine.sort(sort2D); //2. извиквам функцията за сортиране, не става само с една

// console.log(sorted); //3. може направо да се принтира сортирания масив от обекти

return sorted; //4. или да се върне, за допълнителна обработка

function sort2D(a,b) { //1. функцията винаги е с (a,b), нали сортира

let firstCriteria = b.age - a.age; //2. сорт по първи критерий(.age, .length и т.н.)

if (firstCriteria === 0) { //3. ако първият връща, че са равни

return a.name.localeCompare(b.name); //4. връща сортиран по втори критерий и излиза от функцията

} // (не стана с[name]), а с .name

return firstCriteria; //5. връща сортиран по първи критерий и излиза

}

}

sortObject(arrayObj); // arrayObj e масивът за сортиране по два критерия

console.log(arrayObj);

**ASSOCIATIVE ARRAYS – АСОЦИАТИВНИ МАСИВИ**

Обектите играят ролята на вид асоциативни масиви

Не могат да имат само пропърти, без стойност.

Ас. Масиви - приличат на обекти, инициализират се като обекти:

**let assocArr = {** let assocArr = {};

**'one': 1, или** assocArr['one'] = 1;

**'two': 2** assocArr['two'] = 2;

**};**

**Достъпване на стойност (value): assocArr['*key*'] или assocArr.*key***

**Добавяне на стойност от масива**

**към някаква нова променлива: let age = assocArr[key]; или let age = assocArr[key];**

**Итериране/ Обхождане през асоциативен масив – с FOR-IN :**

let assocArr = {};

assocArr['one'] = 1;

assocArr['two'] = 2;

**for(let key in assocArr) {**

**console.log(key + " = " + assocArr[key]);  *-* Ключа(key) се вади динамично (ако например не знаем, че е ‘one’,’two’…)**

**}**

**Превръщане на КЛЮЧОВЕ - В МАСИВ с forEach (за АС. Масив и за ОБЕКТ):**

**Може и за ОБХОЖДАНЕ с forEach:**

**let assocArr = {};**

**assocArr['one'] = 1;**

**assocArr['two'] = 2;**

**assocArr['three'] = 3;**

**Object.keys(assocArr).forEach((i)=> { // ПРИНТИРАНЕ на ас. Масив (има го и по-надолу)  
 console.log(`${i} = ${assocArr[i]}`)); // one = 1 two = 2 и т.н.**

**}**

**КОПИРАНЕ на един ас. Масив в друг:**

**Let animal = {};**

**Let existingAnimal = {  
name: ‘Joe’,**

**Age: 12,**

**Color: ‘brown’,**

**};**

**For(let key in existingAnimal) {**

**(Animal.key – НЕ СЕ ИЗПОЛЗВА ПРИ ОБХОЖДАНЕ С ЦИКЪЛ, защото ще търси ключ с име key)**

**Animal[key] = existingAnimal[key]; - създават се нови стойсности в обекта animal със същите ключове.**

**(сетване на ключ и сотйност = четене на стойност от ключ)**

**}   
Все едно animal[name] = existingAnimal[name];  
 animal[age] = existingAnimal[age];**

**ПРИНТИРАНЕ на ас. Масив с FOR-IN:**

**For (const key in animal) {**

**Console.log(`Key: ${key} = Value: ${animal[key]}`);**

**}**

**ENUMERABLE (МОЖЕ ДА СЕ НОМЕРИРА) – може да се обходи с for-of и с for-in.**

Class Person {

**Constructor(name, age) {**

**This.name = name; // пропъртита на обекта**

**This.age = age;**

**}**

**sayName() { // методи на пропъртита на прототипа на обекта**

**console.log(this.name);**

**}**

}

// ПРИНТИРАНЕ САМО НА КЛЮЧОВЕ(пропъртита) на КЛАС

For(const prop in new Person()) {

Console.log(prop ); // name Прнтира само ключовете

} age  
const p = new Person(‘Joe’,27); // създаване на нова инстанция p на прототипа Person

For (const key in p) {

Console.log(p[key]); // Joe

} 27

//Обект:

Const **animal** = {

Name: ‘Joe’,

Color: ‘black’

};

// ПРИНТИРАНЕ **САМО НА КЛЮЧОВЕ**(пропъртита) на ОБЕКТ

**Object.keys(foo).forEach(key => console.log(key));**  // name color

//Създаваме **ФУНКЦИЯ ЗА ПРИНТИРАНЕ НА ОБЕКТ**

Function logKeyAndValue(key) {

Console.log(`key: ${key} ${this[key]}`);

}

For(const key in **animal**) {

logKeyAndValue.**call(animal, key);** // **Key: name Joe** ${this[key]} **.call** прави **animal** да застава вместо **this**

} // **Key: color black ${animal[key]} call работи с this**

Object.keys(animal) // дава масив от ключовете (на animal) в стринг, които се ползват от функцията -

.forEach(logKeyAndValue, animal) // logKeyAndValue, а на втора позиция – кой е this-а (в случая animal)

}

**Key: name Joe** // Отпечатва пак същото

**Key: color black**

**FOR-IN() , .FOREACH ( (key) => key.asdf ) , OBJECT.KEYS(object) – Обхождат по key**

**Когато ключовете са числа, се получава нещо като автоматично сортиране по възходящ ред.  
 const bar = { 1: 10, 0: 20, 10: 100, 5:2000 }**

**Object.keys(bar).forEach(key => console.log(key)); // 0 1 5 10**

**JSON.stringify(bar) // {“0”:20,”1”:10,”5”:2000,”10”:100} - сортира числата на ключовете**

**Пълнене на ас. Масив С ПРЕЗАПИСВАНЕ НА ДУБЛИРАЩИ СЕ КЛЮЧОВЕ – получава се презаписване по реда на въвеждане, автоматично...**

Const names = [‘Tim’, ‘Peter’, ’Bill’, ‘Tim’];

Const phones = [ ‘023131’, ‘1231231’, ‘01231231’, ‘12312312’];

Function thelephoneBokThem(names,phones) {

Const telephoneBook = {}; // създаваме го, да ни пази keys and values;

For (const index in names) { // НЕ става с for-of, защото после се ползва indexOf, който взема само първия срещнат name

Const name = names[index]; // взимаме името от масива с имената по индекс 0,1,2,3

Const phone = phones[index]; // взимаме телефоните от съответната позиция в масива с номерата 0,1,2,3

telephoneBook[name] = phone; // **Пълним обекта(ас.масив) с имена и номера**

}

**// ДВА ДРУГИ ВАРИАНТА ЗА ПЪЛНЕНЕ НА ОБЕКТА (АС. МАСИВ)**

**//** for (let I = 0; I < names.length; i++) { **// С for цикъл. Най-експлицитния метод**

**//** telephoneBook[names[i]] – phones[i];

**//** }

**//** Object.values(names).forEach( (name, index) => { // **Object.values(names)** – връща масив с тюпъли, на който

**//** telephoneBook[name] = phones[index]; // **.forEach( (name, index) =>** - на всеки тюпъл името(name) и номера(index)

**//** } ); // запълват обекта (асоциативния масив).

For (const name in telephoneBook) {

Const phone = telephoneBook[name]; // не задължително да направим променлива phone за value на key – имената

Console.log(`${name} -> ${phone}`); // принтиране ключ -> сойност на ключа

}

}

Решението, както е в презентацията от лекцията за ас. Масиви:

**function solve(input) { // входът е стрингове с по два елемента, сплитнати по интервал**

**let phonebook = {};  
 for (let string of input) {  
 let tokens = string.split(' ');  
 let name = tokens[0];  
 let number = tokens[1];  
 phonebook[name] = number; // пушване в обекта(ас. масива)  
 }  
 for (let key in phonebook) {  
 console.log(`${key} -> ${phonebook[key]}`);  
 }  
}**

--------------------------------------------------------------------------

**Map()** - **Class**, подобен на Dictionary в Java и C#

Const person = {};

Person.name = ‘Joe’;

**ИЗТРИВАНЕ НА ПРОПЪРТИ: delete person.name;**

**ПРОВЕРКА ДАЛИ ИМА ТАКОВА ПРОПЪРТИ: person.hasOwnProperty(‘name’) //** true, false

**ИЗТРИВАНЕ НА ВСИЧКИ ПРОПЪРТИТА:**

**For (const key in person) {**

**delete person[key];**

**}**

Const myMap = new Map();

myMap**.set**(‘name’, ‘Joe’); // **Задаване** на стойности със **.set** Output: **Map(1) {“name” => “Joe”}**

myMap**.get**(‘name’); // **Връщане** на стойност по ключ **.get**  Output: **“Joe”**

myMap**.has**(‘name’); // **Проверка** дали има пропърти **.has** Output: **true**

myMap**.delete**(‘name’); // **Изтрива** ключ/стойност двойката **.delete**

myMap**.clear**(); // **Изтрива** **всички двойки** ключ/стойност **.clear**

myMap**.size** // Връща число – **дължина на обекта**

myMap**.entries() //** Връща двумерен MapIterator **[ key, value ],** (който може да бъде обхождан (**enumerable**) )?

**let entries = Array.form ( myMap.entries()** ); // [ [ ‘name’, ‘Joe’ ] , [ ‘age’, 27 ] ] **За да се обхождат, се каства с Array.form()**

myMap**.keys() //** Връща MapIterator от ключове ( MapIterator {1, 2} ) При обект: Object.keys(Obj);

**let keys = Array.from ( myMap.keys() );** // [ ‘name’, ’age’ ]

myMap**.values() //** Връща MapIterator от **стойности**, (който може да бъде обхождан (**enumerable**))?

**let values = Array.from ( myMap.values() );** // [ ‘Joe’, ’27’ ]

**let values = [ …myMap.values() ] – друг начин за превръщане в масив!**

**Map конструктора приема масив от тюпъли:**

Const map = new Map( [ [‘foo’, ‘bar’ ] , [ ‘bar’, ‘baz’ ] ] ); // { “foo” => “bar” , “bar” =. “baz” }

**For ( const [key, value] of map ) { // Деструктуриране,** когато се знае форматът

**Console.log(key, value);** // foo bar

**}** // bar baz

**For ( const [ , value] of map ) { //** Деструктуриране, **с ползване на част от данните (броя на запетаите е броя на пропуснатите)**

**Console.log(value);** // bar

**}** // baz

**Преобразуване от Обект в Map()** става, като първо обектът се преобразува в масив:  
const map = new Map ( Object.entries ( {a: 1, b: 2} ) ) // Map(2) {“a” => 1, “b” => 2}

**ПРИНТИРАНЕ НА MAP** става с ключ или стойност в FOR-OF

**let iterable = Array.from(phonebookMap.entries()); // Превръщане в масив**

**for(let kvp of iterable) { // for-of**

**let name = kvp[0];**

**let number = kvp[1];**

**console.log(`${name} => ${number}`);**

**}**

**ЗА ИЗПИТА:** Ако се иска рещението да е с Мар, може да се работи с обект и накрая да се върне return new Map()

**ДОБАВЯНЕ НА СТОЙНОСТ** към съществуващ ключ с използване на Map

**function storeProductsInfo(products) {**

**const info = products.split(' ');**

**const map = new Map();**

**for (let i = 0; i < info.length; i += 2) {**

**const product = info[i];**

**const quantity = parseInt(info[i + 1]);**

**if (map.has(product)) {**

**const newQuantity = map.get(product) + quantity;**

**map.set(product, newQuantity);**

**} else {**

**map.set(product, quantity);**

**}**

**}**

**//Принтиране - правим масив и го принтираме**

**let array = [...map];**

**for (const line of array) {**

**console.log(`${line[0]} -> ${line[1]}`);**

**}**

**// console.log(map); // Принтиране на суровия map**

**// return map; // връщане на map**

**}**

**storeProductsInfo('tomatoes 10 coffee 5 olives 100 coffee 40');**

**ДОБАВЯНЕ НА СТОЙНОСТ** към съществуващ ключ с използване на Object

**function withObject(input) {**

**let products = {};**

***for (const i of input) {***

***let line = i.split(' ');***

***let product = line[0];***

***let quantity = parseInt(line[1]);***

***if (products.hasOwnProperty(product)) { // Ако има продукт с такова име***

***products[product] += quantity; // да се добави към неговата стойност***

***} else { // в противен случай***

***products[product] = quantity; // да се създаде ново пропърти 'product': 'quantity'***

***}***

***}***

**// ПРИНТИРАНЕ НА ОБЕКТ**

**Object.entries(products).forEach((product) => console.log(`${product[0]} -> ${product[1]}`));**

**// console.log(products);**

**}**

**withObject(['tomatoes 10', 'coffee 5', 'olives 100', 'coffee 40']);**

**Сортиране на Map по стойност и по ключ:**

**function sort(array) {**

**let map = new Map();**

**map.set("one", 1);**

**map.set("eight", 8);**

**map.set("two", 2);**

**// СОРТИРАНЕ ПО СТОЙНОСТ**

**let sortedValues = Array.from(map.entries()) // правим масив от тюпъли и сортираме**

**.sort((a, b) => a[1] - b[1]); // по [1] - втория елемент (стойност)**

**// СОРТИРАНЕ ПО КЛЮЧ**

**let sortedKeys = Array.from(map.entries())**

**.sort((a, b) => a[0].localeCompare(b[0]));**

**// ПРИНТИРАНЕ В КОНЗОЛАТА**

**for (let kvp of sortedValues) {**

**console.log(`${kvp[0]} -> ${kvp[1]}`);**

**}**

**for (let kvp of sortedKeys) {**

**console.log(`${kvp[0]} -> ${kvp[1]}`);**

**}**

**}**

**sort([['one', 1], ['eight', 8], ['two', 2]]);**

**function anotherSort() {**

**const data = [['one', 1], ['three', 3], ['minusFive', -5]];**

**const map = new Map(data);**

**// взимаме само стойностите, като превръщаме map в масив([...map]) и**

**// от всеки тюпъл(.map()) взимаме само вторият елемент - стойността**

**const valuesOnly = [...map].map(tuple =>/\*no tuple[0]\*/ tuple[1]);**

**valuesOnly.sort((a, b) => a - b); // [ -5, 1, 3 ] сортиране по възходящ ред**

**valuesOnly.sort((a, b) => b - a); // [ 3, 1, -5 ] сортиране по низходящ ред**

**console.log(valuesOnly);**

**}**

**anotherSort()**

**function anotherSortShort() {**

**const data = [['one', 1], ['three', 3], ['minusFive', -5]];**

**const map = new Map(data);**

**let sortedValues = [...map] // Сортиране по стойност**

**.map(tuple => tuple[1])**

**.sort((a, b) => b - a);**

**console.log(sortedValues); // Сортиране по ключ**

**let sortedKeys = [...map]**

**.map(tuple => tuple[0])**

**.sort((a, b) => b.localeCompare(a));**

**console.log(sortedKeys);**

**}**

**anotherSortShort()**

**СОРТИРАНЕ НА ОБЕКТИ**

**// СОРТИРАНЕ НА ОБЕКТИ (ПО СТОЙНОСТ) С ЕДНАКВИ КЛЮЧОВЕ**

**function sortObject() {**

**const data = [['over 18', { age: 24 }], ['under 18', { age: 12 }]];**

**let sorted = [...new Map(data)] // [...спредва се] до масив от тюпъли**

**.map(tuple => tuple[1]) // взимаме вторите им елементи[1] - {обектите}**

**.sort((a,b) => b.age - a.age); // сортираме по ключ .age**

**console.log(sorted); // принтираме масив от сортирани обекти**

**}**

**sortObject()**

**// Сортиране на два обекта (по стойност) с различни ключове**

**function sortObj() {**

**const data = [['over 18', { john: 24 }], ['under 18', { simon: 12 }]];**

**let sorted = [...new Map(data)] // [...спредва се] до масив от тюпъли**

**.map(tuple => tuple[1]) // взимаме вторите им елементи[1] - {обектите}**

**.sort((a,b) => b.john - a.simon); // сортираме по ключ .age**

**console.log(sorted); // принтира масив от сортирани обекти**

**}**

**sortObj()**

We can use both **Arrays** and **Maps** to store **key-value** pairs

**Maps** are a better way to do it because: They are **iterable,** They have **size property**

They are better for **adding** and **deleting** many **key-value** pairs

**SET() – запазва само уникалните стойности (примитивни или обекти), може да итерира през елементите**

console.log**(set.has(1)); *// Expected output: true***

Обаче не може да сравнява обекти дали са еднакви и да изключва повтарящи се. Затова трябва да се преобразуват в масиви.

let objects = [{ a: 1 }, { a: 1 }, { a: 1 }];

let obj = objects.map(object => JSON.stringify(object)); //5. масивът е стрингифайнат

console.log(obj);

let set = new Set(obj); //6. Set { '{"a":1}' } повтарящите елементи са елиминирани

console.log(set);

**TEXT PROCESSING**

<https://greensock.com/gsap> - анимации js

<https://eslint.org/> - плъгин, който следи написания код да спазва „добрите практки“. Масово използван.

Стринговете не могат да бъдат променяни. Ако се сплитне на ЧАРОВЕ и се ползва индекс, тогава става

“word” =>[ “w”,”o”,”r”,”d”]

.trim() - премахва спейсове пред и зад стринга

* **Str.substr (**startIndex, length**) // изрязва парче (старт, дължина)**
* **Str.substring (**startIndex, endIndex**?) // изрязва парче (старт, край?) без последната буква**
* **replace(**search, replacement**) // реплейсва първият срещнат(при RegExp – всички) стринг с посочения**

**JavaScript Web Scraping – за взимане на данни от друг сайт!!!**

.forEach() не работи на стрингове, защото това е метод на прототипа на масивите. Стринговете не са typeOf array.

.toLowerCase() – помага много

**‘\*’.repeat(5) – повтаря определен брой пъти някакъв стринг/чар \*\*\*\*\***

**.trim() - премахва whitespace спейсове и табулации от двете страни (‚ John ‘) => ‘John’**

**.trimStart(), .trimEnd() - чистят само в началото или само в края**

**console.log(text.startsWith('My')); *// Expected output: true***

**console.log(text.endsWith('John')); *// Expected output: true***

// Например, да проверим дали изречение започва с въпорс

const questionWords = ['what', 'why', 'when', 'who', 'should', 'how', 'is', 'can'];

function isQuestion(str) {

let isIt = false;

str = str.trim(); // за да премахнем пространствата преди и след изречението

// допълнителна функция, за главна буква:

function capitalizeFirstLetter(string) {

// връща стринг, който взима първата буква, прави я главна и +залепя останалата част от думата отзад

return string.charAt(0).toUpperCase() + string.slice(1);

}

// въпреки че може да се направи .toLowerCase(). Тук съм сложил и двете :)

questionWords.forEach(word => {

if ((str.toLowerCase().startsWith(word) || str.startsWith(capitalizeFirstLetter(word)))

&& str.endsWith('?')) {

isIt = true;

}

});

return isIt;

}

console.log(isQuestion(' When did you eat ? '))

**padStart()**

**console.log('L'.padStart(5, 'X')); // XXXXXL**

**// става за добавяне на падинг(идентация) в текстообработка**

**31-REGULAR EXPRESSIONS – спрямо ASCII**

\w – буква**[a-zA-Z0-9\_]**, \w+ - буква \w{1,3} – от 1 до 3 думи \W – всичко, което не е буква, обратното на \w, както ^

\d – цифра**[0-9],** \d+ - цифри \d{5} - 5 цифри \s – интервал **\S+ – ВЯКАКВИ ВЪЗМОЖНИ без интервали**

[0-9] – числа [a-z] – малки букви [A-Z] – големи букви [^abc] ^ - знак за отрицание

[a-zA-z0-9] – всички малки, големи букви и цифри

[a] – всяко а [abc] всяко a, b , c

[^a] - без А [^abc] - без a, b, c а\* - 0 или повече пъти ‚а‘, а? 0 или 1 път ‚а‘

**\d{1,2}-\w{3}-\d{4}** - (**d)d – mmm-yyyy** **Eдна или две цифри** за дни, три букви за месец и 4 цифри за година

**\d{1,2}-([A-Z])\w{2}-\d{4}** - (**d)**d – **M**mm-yyyy Същото, с главна буква за месец

**\+\d+** +35989854948 - eескейпнат плюс и числа

**(Hi|Hello), (Pesho|Gosho)** - Hi, Pesho Hello, Pesho Hi, Gosho Hello, Gosho

**(Hi|Hello)(Pesho|Gosho)** - HiPesho HelloPesho HiGosho HelloGosho

**(Hi|HelloPesho|Gosho)** - Скобите се използват за отделяне, за да не се залепват експрешъните, и за рефериране  
При рефериране **отварящата скоба** определя номера на референцията!!!

**(?:Hi|Hello), (?:Pesho|Gosho)** - най-експлицитно, заради js

**?:** - non capturing group. Нов начин на запис е с ключове-имена **(?<day>\d{2})-(?<month>\w{3})-(?<year>\d{4})**/g

**^\w+@\w+.\w+$** - [mar\_ilije@hotmail.com](mailto:mar_ilije@hotmail.com) ^a$ - обхваща стринга с ^(начало) и $(край)

.\* - 0 символа или каквото и да било

Задача за прихващане на <b>ВАЛИДНИ</b> тагове (отварящ да отговаря на затварящ таг)

<(\w+).\*>.\*<\/\1>:   
< > - обозначаване на таг <p>;

<( )> - обозначаване на кепчър груп (), за рефериране към затварящ таг;

<( \w+).\*> - **\w+** е за името на тага **.\*** - ако има **a** href=’www.some.bg’

<(\w+).\*>.\* - **.\*** - текст на тага

<(\w+).\*>.\*<\/\1> - **<\/\1>** - затварящ таг </p> с ескейпване на / и рефериране към прочетеното от референтните скоби \1

**Character classes Класове на знаците**

. any character except newline **всеки символ**, освен нов ред

\w \d \s word, digit, whitespace дума, цифра, празно пространство

\W \D \S not word, digit, whitespace не дума, цифра, празно пространство

[abc] any of a, b, or c всяко от a, b или c

[^abc] not a, b, or c не a, b или c

[a-g] character between a & g знак между a & g

**Anchors Котвите**

^abc$ start / end of the string начало / край на низ

\b \B word, not-word boundary обозначаване на граница на дума(начало или край[както е <spa>]), не-граница на дума

**Escaped characters Извадени знаци**

\. \\* \\ \/ escaped special characters ескейпване на специални знаци. Ако са в [.?!$] нямат нужда от ескейпване, [] ги ескейпва

\t \n \r tab, linefeed, carriage return табулация, подаване на ред, връщане на каретата

**Groups & Lookaround Групи и изглед наоколо**

(abc) capture group група за улавяне – **използва се за рефериране по пореден номер на използване.**

\1 backreference to group #1 обратна връзка към група # 1 - **реферира към (abc) но взима мапнатата стойност**

(?:abc) non-capturing group група, която не се показва в резултата (например при exec)

(?=abc) positive lookahead поглед пред патерна ни. (Преглежда дали има нещо **след** патерна ни,без да е част от мача)

(?!abc) negative lookahead отрицателен изглед

**Quantifiers & Alternation Квантори и алтернатива**

a\* a+ a? 0 or more, 1 or more, 0 or 1 0 или повече, 1 или повече, 0 или 1

a{5} a{2,} exactly five, two or more точно пет, две или повече

a{1,3} between one & three между едно и три

a+? a{2,}? match as few as possible съвпадат възможно най-малко

ab|cd match ab or cd съответства на ab или cd

| **Regex Method** | **Description** |
| --- | --- |
|  | **Exec и test се извикват върху pattern-a: let pattern = /\d{4}/gi;** |
| [exec](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/exec) | **A RegExp method** that executes a search for a match in a string. It returns an array of information or null on a mismatch. **pattern.exec(string)** |
| [test](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/test) | **A RegExp method** that tests for a match in a string. It returns true or false. **pattern.test(string)** |
|  | **Останалите са стрингови и се извикват върху текста: let text = "Lorem Ipsum…“** |
| [match](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/match) | **A String method** that returns an array containing all of the matches, including capturing groups, or null if no match is found. **text.match(pattern)** |
| [matchAll](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/matchAll) | **A String method** that returns an iterator containing all of the matches, including capturing groups. |
| [search](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/search) | **A String method** that tests for a match in a string. It returns the index of the match, or -1 if the search fails. |
| [replace](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace) | **A String method** that executes a search for a match in a string, and replaces the matched substring with a replacement substring. |
| [split](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/split) | **A String method** that uses a regular expression or a fixed string to break a string into an array of substrings. |

Позитивен и негативен луук бихайнд   
(?<=a)b - Преглежда дали има нещо пред патерна ни,без да е част от мачнатия стринг. Ще мачне **b** в a**b**c, но не в c**b**a

(?<!a)b – Обратното. Ще мачне **b** в c**b**a, но не и в a**b**c.

const pattern = / \%(?<customer>\b[A-Z][a-z]+)\%(\w\*[^\||\$|\%|\.]\*)<(?<product>\w+)>(\w\*[^\$|\%|\.]\*) \|(?<quantity>\d+)\|[^\||\$|\%|\.|\d]\*[A-Za-z]\*(?<price>\d+.\*\d+)\$ /gm; // трябва да е вътре

**Each valid order** should have a **customer, product, count and a price:**

Valid customer's name should be **surrounded by '%'** and must **start with a capital letter**, followed by **lower-case letters** - \%(?<customer>\b[A-Z][a-z]+)\%

(\w\*[^\||\$|\%|\.]\*) - Between each part there can be other symbols, a mi (**'|', '$', '%' and '.'**)

Valid product **contains any word character** and must be **surrounded by '<' and '>'** - <(?<product>\w+)>

* Valid count is an **integer**, **surrounded by '|' -** \|(?<quantity>\d+)\|

[^\||\$|\%|\.|\d]\*[A-Za-z]\*

* Valid price is any **real number followed by '$' -** (?<price>\d+.\*\d+)\$