**BOM**

**Introduction :**

**B**rowser **O**bject **M**odel ‘**BOM**’ allows JavaScript to « talk to « the browser.

**1- Alert, Confirm, Promt :**

**Alert()**

First there no diffrence between :

window.alert("Hello 1");

alert("Hello 2");

this.alert("Hello 3");

// this = window Object

**Note :**

The alert box takes the focus away from the current window, and forces the user to read the message.

It’s not recomanded to use in our time, you have other good choices or you can do it by yourself you can do a good one by you knowledge and better in optimasation, you can use some libraries like « sweet alert « .

**confirm()**

The **confirm()** method displays a dialog box with a message, and OK button, and a Cancel button.

The **confirm()** method returns true if the used clicked « ok », otherwise false

let confirmMsg = confirm("Are you Sure ?");

console.log(confirmMsg === true ? "Item Deleted" : "Item has not deleted");

**Note :**

The same (alert , confirm , prompt)

**Prompt()**

The **prompt()** method displays a dialog box that prompts (drives) the user for input.

The **prompt()** method returns the input value if the user clicks «Ok », otherwise it returns null

let promptMsg = prompt("What's your name ?", "Harry Potter");

console.log(`Hello ${promptMsg}`);

**syntax :**

prompt(text,defaultText)

**text :** optional, the text to display in the dialog box

**defaultText :** optional, the default input text

**2- setTimeOut , clearTimeout() :**

The **setTimeout()** method calls a function after a number of miliseconds.

1 second = 1000miliseconds.

setTimeout(function () {

  console.log("Msg");

}, 1000);

Let’s make an external function with param

setTimeout(sayMsg, 1000);

function sayMsg(user) {

  console.log(`Hello ${user}`);

}

To be honest you can’t add any param in () because the building of setTimeout() doesn’t allow to add param in () , but there a special please for params :

**setTimeout(function(),miliseconds,param1,param 2 , ……..)**

parameters : optional (just if you need them)

setTimeout(sayMsg, 1000, "Bilal", 17);

function sayMsg(user, age) {

  console.log(`Hello ${user},His age is ${age}`);

}

**Note :**

The **setTimeout()** is executed only once.

Use the **clearTimeout()** method prevent the functin from starting

To clear a timeout, use the id returnd from **setTimeout(**).

let timeout = setTimeout(sayMsg, 2000, "Bilal", 17);

Then you can stop the exection by calling **clearTimeout()**

clearTimeout(timeout);

let timeout = setTimeout(sayMsg, 2000, "Bilal", 17);

function sayMsg(user, age) {

  console.log(`Hello ${user},His age is ${age}`);

}

document.getElementById("button").onclick = () => clearTimeout(timeout);

or you can use the rank of timer :

console.log(timeout); // 1

as we know he is the first one ;

clearTimeout(1) ;

**3- setInterval() , clearInterval() :**

The **setInterval()** method calls a function at specified intervals (in miliseconds) .

The **setInterval()** method continues calling the function until **clearInterval()** is called, or window is closed.

let p = document.querySelector("p");

p.style.cssText = "font-size:300px;text-align:center;color:olive";

setInterval(function () {

  p.textContent = p.textContent - 1;

  if (p.textContent === "0") {

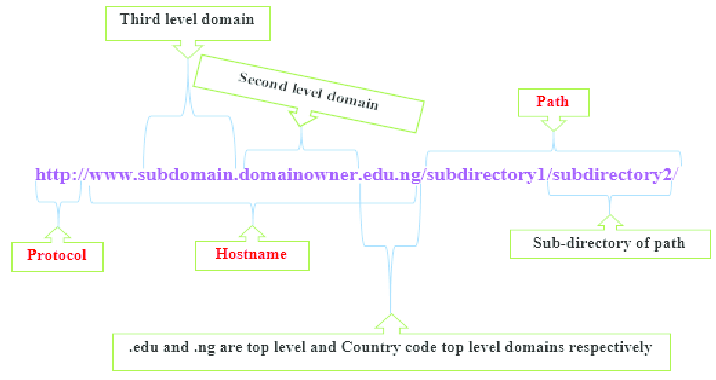
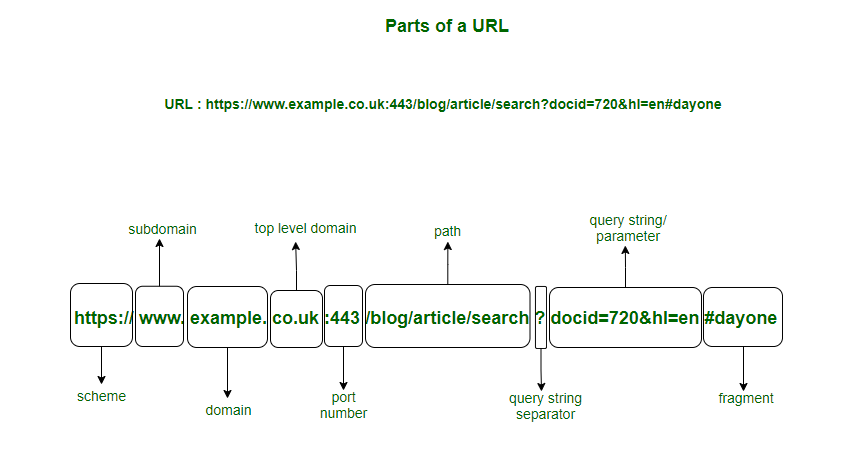
    p.textContent = "Over";

    clearInterval(1);

  }

}, 1000);

**4-Window Location:**

The **Window.location** object can be used o get the current page address and to redirect the borowser to a new page.

The **location.href** property returns the URL of the current Page

console.log(location.href);

or you can set your hred but the last page will be saved in History

location.href = "https://www.google.com";

The **location.hostname** property returns the name of the internet Host

console.log(location.hostname); // www.researchgate.net

The **location.path** property returns the path

console.log(location.pathname); // /js/js\_window\_location.asp

The **location.protocol** property returns the web protocol of the page

console.log(location.protocol); //http:

The **location.port** property returns the number of the internet host port.

Most browsers will not dispaly default port numbers (80 for http and 443 for https ).

The **location.reload()** method reloads the current document.

The **reload()** method does the same as the reaload button in you browser.

setTimeout(function () {

  location.reload();

}, 2000);

The **location.replace()** method replaces the current document with a new one .

**Parameter :** *required.*

*The URL of a page to navigate to.*

The difference between **assign()** and **replace()** :

**Replace()** removes the current URL from the document history.

With **replace()** it is ***not possible*** to use ‘back’ to navigate back to the original document.

**5-Window Open & Close:**

The **open ()** method opens a new browser window, or a new tab, depending on you browser settings and the parameter values.

**Syntax :**

**window.open(*URL, name, specs, replace*)**

name : \_blank or \_self

setTimeout(function () {

  open(

    "https://www.facebook.com",

    "\_blank",

    "width=400,height=300,top=400,left=400"

  );

}, 1000);

**6-Window History:**

The **window.history** object contains the browser history.

* **history.back() - same as clicking back in the browser**
* **history.forward() - same as clicking forward in the browser**

the **history.length** returns the number of URLs in the history list of the current browser window + the window that are you in.

This property is useful to find out how many pages the user has visited in the current browsing session.

The **history.go()** method used to move in history list .

*History.go(0)*reloads the page.

*History.go(-1*)is the same as history.back()

*History.go(-2)*for two stpes backward in history list.

*History.go(1)* is the same as history.forward()

I think you got it

**7- Scroll,ScrollTo,ScrollBy,Focus,Print,Stop:**

The **stop()** method stops window loading.

The **stop()** method is the as clicking stop in the browser.

Note :

The **stop()** method can be used to stop loading an image if it takes too long.

The **print()** method prints the contents of the current window

The **print()** method opens the print dialog box,which lets the user to select preferred printing options.

let btn = document.querySelector("#btn");

btn.onclick = () => print();

The **focus()** method sets focus to a window.

The **blur()** method removes focus from a window.

Note :

The **focus()** method makes a request to bring a window to the front.

It may not work as you expect, due to different user settings.

The **scrollTo()** method scrolls the document to specified coordinates.

window.scrollTo(500, 500);

setTimeout(function () {

  window.scrollTo({ left: 500, top: 700, behavior: "smooth" });

}, 1000);

The **scrollBy()** method scrolls the document by the spcified number of pixels.

scrollBy(100, 100);

the **scrollX** property returns the pixels a document has scrolled for the upper left corner of the window.

The **scrollX** is read-only.

The **scrollX** property  is equal to the **pageXoffset** property.

The same for **scrollY**.

setInterval(function () {

  console.log(scrollX, scrollY);

}, 1000);

Practice :

let btn = document.getElementById("btn");

btn.style.cssText =

  "position:fixed;bottom:50px;right:20px;background:dodgerblue;color:white;border-radius:5px;font-weight:bold;padding:10px;cursor:pointer;display:none";

window.onscroll = function () {

  console.log(scrollY);

  if (scrollY > 1000) {

    btn.style.display = "block";

  } else {

    btn.style.display = "none";

  }

};

btn.onclick = () =>

  scrollTo({

    top: 0,

    behavior: "smooth",

  });

The **localStorage** object allows you to save Key/Value pairs in the browser.

**Note :**

The **localStorage** object stores data with no expiration date.

The data in not deleted when the browser is closed, and are available for future sessions.

console.log(window.localStorage);

// set

localStorage.setItem("Background", "dodgerBlue");

localStorage.color = "white";

localStorage["fontSize"] = "20px";

// get

console.log(localStorage.getItem("Background"));

console.log(localStorage.color);

console.log(localStorage["fontSize"]);

// get Key "to get the key an index"

console.log(localStorage.key(1)); // color

// remove

localStorage.removeItem("fontSize");

// Remove All "Clear"

localStorage.clear();

console.log(localStorage);

Practice :

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <link rel="stylesheet" href="main.css" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Js test</title>

    <style>

      .remote {

        width: 500px;

        background: #c0c0c06b;

        height: 80px;

        display: flex;

        justify-content: space-evenly;

        align-items: center;

        padding: 5px;

        margin: 50px auto;

        border-radius: 5px;

        display: flex;

      }

      .remote span {

        width: 50px;

        height: 50px;

        cursor: pointer;

      }

      .red {

        background: #e50000;

      }

      .green {

        background: #90ee90;

      }

      .purple {

        background: #cbc3e3;

      }

      .gray {

        background: gray;

      }

      .remote .active {

        transition: 0.5s;

        transform: scale(1.3);

        border: 3px solid white;

        border-radius: 5px;

      }

      .screen {

        width: 500px;

        height: 300px;

        margin: auto;

        border-radius: 5px;

      }

    </style>

  </head>

  <body>

    <!-- ////////////// -->

    <div class="remote">

      <span class="red" data-color="#e50000"></span>

      <span class="green" data-color="#90ee90"></span>

      <span class="purple" data-color="#cbc3e3"></span

      ><span class="gray" data-color="gray"></span>

    </div>

    <div class="screen"></div>

    <!-- ////////////// -->

    <script src="main.js"></script>

  </body>

</html>

let spans = document.querySelectorAll("span");

let screen = document.querySelector(".screen");

if (localStorage.getItem("active")) {

  let active = document.querySelector(`.${localStorage.getItem("active")}`);

  active.classList.add("active");

  screen.style.backgroundColor = localStorage.getItem("BackgroundColor");

} else {

  let active = document.querySelector(".red");

  active.classList.add("active");

  screen.style.backgroundColor = active.getAttribute("data-color");

}

spans.forEach((element) => {

  element.addEventListener("click", (e) => {

    // Remove active class from all spans

    spans.forEach((span) => span.classList.remove("active"));

    // add active class to clicked span

    e.target.classList.add("active");

    // add BackgroundColor to local Storage

    localStorage.setItem(

      "BackgroundColor",

      e.target.getAttribute("data-color")

    );

    localStorage.setItem("active", e.target.classList[0]);

    // give background to screen based on BackgroundColor saved in localStorage

    screen.style.backgroundColor = localStorage.getItem("BackgroundColor");

  });

});

The **sessionStorage** object let you store Key/Value pairs in the browser.

**Note :**

The **sessionStorage** object stores data for only one session.

(the data is deleted when the browser is closed).

**sessionStorage** and **localStorage** has the same methods.

Info :

New Tab = New Session (جلسة )

Duplicate Tap = Copy Session

New Tap with same url = new session

**Challenge Note :**

When sending data to a web server, the data has to be a string.

Convert a js object into a string with JSON.stringify().

It is also possible to stringify js arrays

myobj = {

  name: "bilal",

  age: 17,

};

console.log(myobj); // obj

console.log(JSON.stringify(myobj)); // {"name":"bilal","age":17} "as a string "

when storing data, the data has to be a certain format, and regardless of where you choose to store it, the text is always one of legal formats.

localStorage.setItem("myObject", JSON.stringify(myobj));

so the previous code makes us able to store an object in Lc

AND FOR make it back for origin format, use JSON.parse().

when receving data from a web serveur, the data always a string

Parse the data with JSON.parse(). And the data becomes to a js object {original format}

console.log(localStorage.getItem("myObject"));

console.log(JSON.parse(localStorage.getItem("myObject")));

**Destructing**

To illustrate destructing, we’ll make a sandwich.do you take everything out of the refrigerator to make your sandwich ? No, you only take out the items you would like to use on your sandwich.

Destructing makes it easy to extract only what is needed.

**Destructing Arrays :**

Here is the old way of assigning array items to a variable :

**Before :**

const vehicles = ["mustang", "f-150", "expedition"];

// old way

const car = vehicles[0];

const truck = vehicles[1];

const suv = vehicles[2];

Here is the new way of assigning array items to a variable :

**With Destructuring :**

const vehicles = ["mustang", "f-150", "expedition"];

const [car, truck, suv] = vehicles;

const [car = "dacia", truck, suv, r = "Suzuki"] = vehicles;

// the default value of car is "dacia", it's works like

// an alternate value

**example :**

let myFriends = [

  "Ahmed",

  "Sayed",

  "Ali",

  ["Shady", "Amr", ["Mohamed", "Gamal"]],

];

let [, , , [a, , [, b]]] = myFriends;

console.log(a); // Shady

console.log(b); // Gamal

**Swap Variables (before & after) :**

// Before

let book = "Video";

let video = "Book";

console.log(book, video); // Video Book

let a = book;

book = video;

video = a;

console.log(book, video); // Book Video

/ with Destructuring

let book = "Video";

let video = "Book";

console.log(book, video); // Video Book

[book, video] = [video, book];

console.log(book, video); // Book Video

**Destructing Objects :**

This the old way of storing object properties in variables :

const user = {

  theName: "Bilal",

  theAge: 17,

  theTitle: "Dev",

  theCountry: "Morroco",

};

console.log(user.theName);

let theName = user.theName;

let theAge = user.theAge;

let theTitle = user.theTitle;

let theCountry = user.theCountry;

console.log(theName, theAge, theCountry, theTitle);

**with destructuring :**

// with Destructuring

const user = {

  theName: "Bilal",

  theAge: 17,

  theTitle: "Dev",

  theCountry: "Morroco",

};

const { theName, theCountry } = user;

console.log(theName, theCountry); /// Bilal Morroco

**Note : the object properties do not have to be declared in a specific order.**

**It isn’t like array, now you have properties name, so to extract the needed property you have just to declare its name.**

**-- Hey you can rename the varaibles :**

const { theName: name, theCountry: country } = user;

console.log(name, country); /// Bilal Morroco

**for object also there is default value :**

const user = {

  theName: "Bilal",

  theAge: 17,

  theTitle: "Dev",

  theCountry: "Morroco",

  theColor: "Black",

};

const { theName: name, theCountry: country, theColor = "Red" } = user;

// default value of theColor is red

console.log(theColor); /// Black

**but the question here, that if i want to rename theColor :**

const { theName: name, theCountry: country, theColor: color = "Red" } = user;

**nested object :**

const user = {

  theName: "Bilal",

  theAge: 17,

  theTitle: "Dev",

  theCountry: "Morroco",

  theColor: "Black",

  skills: {

    html: 70,

    css: 80,

  },

};

const {

  theName: name,

  theCountry: country,

  theColor: color = "Red",

  skills: { html: h, css: c },

} = user;

console.log(`My HTML skill Progress is ${h},My CSS skill Progress is ${c}`);

**take a look :**

const {

  theName: name,

  theCountry: country,

  theColor: color = "Red",

  skills,

} = user;

console.log(

  `My HTML skill Progress is ${skills.html},My CSS skill Progress is ${skills.css}`

);

**Destruction for skills**

const { html: skillOne, css: skillTwo } = user.skills;

**Destruction Function Parameters**

**Old : no destructuring**

function showDetails(obj) {

  console.log(`Your Name Is ${obj.theName}`);

  console.log(`Your Age Is ${obj.theAge}`);

  console.log(`Your CSS Skill progress is ${obj.skills.css}`);

}

**With obj destructuring**

function showDetails({ theName, theAge, skills: { css } } = user) {

  console.log(`Your Name Is ${theName}`);

  console.log(`Your Age Is ${theAge}`);

  console.log(`Your CSS Skill progress is ${css}`);

}

**Destruction Mixed Content**

const user = {

  theName: "Bilal",

  theAge: 17,

  skills: ["HTML", "CSS", "JavaScript"],

  addresses: {

    egypt: "Cairo",

    Ksa: "Riyadh",

  },

};

const {

  theName: n,

  theAge: a,

  skills: [one, two, three],

  addresses: { egypt: e },

} = user;

console.log(`Your name is ${n}, Your Age is ${a}`); // Your name is Bilal, Your Age is 17

console.log(`Your Skills is ${one}, ${two}`); // Your Skills is HTML, CSS

console.log(`You Live in ${e}`); // You Live in Cairo

**Set Data Types & Methodes**

A JavaScript Set is A collection of **unique Values.**

Each Value can only occur once is a set.

A Set can hold any values of any data type.

**How To Create a Set ?**

* Passing an Array to new Set ().
* Create a new set and use add() to add values .

let myData = [1, 1, 1, 2, 3, 1, 2];

let myUniqueData = new Set(myData);

console.log(myData); // [1, 1, 1, 2, 3, 1, 2]

console.log(myUniqueData); // Set(3) {1, 2, 3}

2 nd method :

let myUniqueData = new Set();

myUniqueData.add("a");

myUniqueData.add(2);

myUniqueData.add(false);

or

let myUniqueData = new Set().add(1).add(2).add(1).add("h");

**Set Methods :**

***new Set () :*** creates a new Set

***add() :*** Adds a new element to the set

***delete() :*** Removes an element from a Set

***has() :*** returns true if a value exists

***clear() :*** Removes All elements from a Set

**Set Property :**

***size :*** returns the number element in a set (as length in arrays)

myUniqueData.delete(1);

console.log(myUniqueData); // Set(2) {2, 'h'}

console.log(myUniqueData.size); // 2

console.log(myUniqueData.has(2)); // true

myUniqueData.clear();

console.log(myUniqueData.size); // 0

Note :

Delete method return ‘true’ if the element exists and it delete it , and return ‘false’ if the elment doesn’t exist and its sure it will not delete it .

**Values ( )** : Returns an Iterator with all the values in a set .

**Keys () :** the Same as Values()

let myData = [1, 1, 1, 2, 3, 1, 2];

let myUniqueData = new Set(myData);

console.log(myUniqueData); // Set(3) {1, 2, 3}

console.log("Size of Elements Inside Set Is : " + myUniqueData.size);

// Size of Elements Inside Set Is : 3

let iterator = myUniqueData.values();

console.log(iterator.next().value); // 1

console.log(iterator.next().value); // 2

console.log(iterator.next().value); // 3

console.log(iterator.next()); //{value: undefined, done: true}

**values()** returns object with two properties, first named value,second named done, it’s contain a boolean value : true or false ;

the first iteration :

console.log(iterator.next());

//  {value: 1, done: false}

true mean that iteration is done, with wich it loop on all elements, false mean the the process in not yet done.

**forEach ( )** : Invokes a callBack for each element.

myUniqueData.forEach((el) => console.log(el));

// 1

// 2

// 3

WeakSet

let myWeakSet = new WeakSet([{ A: 1, B: 2 }]);

console.log(myWeakSet);

you can’t print weakset elements.

WeakSet is a set, with some small differneces.

The main differences to the set are :

WeakSet is collections of objects only they cannot contain arbitriry values of any type, as set can.

The WeakSet is weak, meaning ,,,,,,, (not clear as well)

Note :

If you don’t want enumaration of set and you have gurantee the set elements will be objects then you should use weakSet.

As we sad there is no size property and there is no clear, keys, values and Entries methods + you can’t use forEach .

**Map**

A Map holds key-value paires where the keys can be any datatype.

A map has a property that represent the size of the map unlike obj that there is no size property

console.log(Object.keys(myObject).length); // to detect obj length

**Map** vs Obj

-- Maps only contain things you put in them -- Objects have prototype which contains more other keys (Bypass with Object.create(null))

-- map has size property unlike obj

-- in obj you can not directly loop but you need to use object.keys()

-- map are has more performance in adding or removing data comaping to map .

At conclusion

**Map is prefered when :**

\* you’re working with big data load

\* you keys are not just string and symbols

**Object is prefered when :**

\* Excellent for quik and small data exchange .

**Map Methods :**

|  |  |
| --- | --- |
| set () | Sets the value for a key a Map |
| get () | Gets the value for a key in a Map |
| clear () | Removes all the elements from a Map |
| delete () | Removes a Map elements specified by a key |
| has () | Returnes true if a key exists in a Map |
| forEach () | Invokes a callback for each key/ value pair in a Map |
| keys () / values() | returns an iterator object of the values in a Map |

myMap.set(110, "heekjk");

myMap.set(() => console.log("hello world"), 99);

console.log(myMap); // Map(2) {110 => 'heekjk', ƒ => 99}

**map vs WeakMap**

first thing we gonna descover what is garbage collection (GC) :

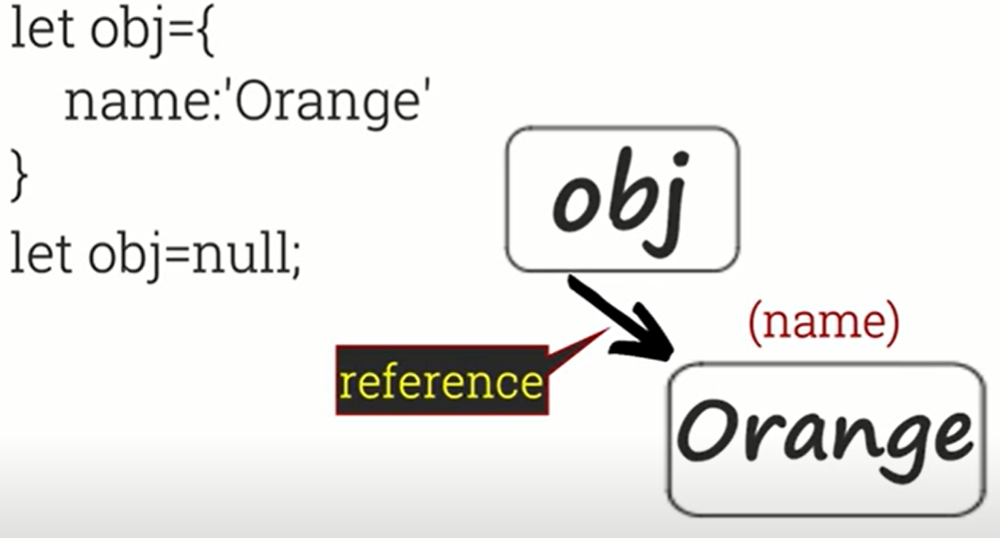
\* Garbage collection is done auto in Js

\* As long as refrences exist, there will no be any cleaning or GC

\* If a location is unrecheable then that will be collected as garbage

What mean the principe of unrecheable ?

Unrecheable mean that the Data has no refrence to it, and as we know we can’t access any data without a refrence to it, that drive us to it.



As the img refere :

\* obj=null remove the link or the refrence between obj and var

So var we be unrecheable so will be GC.

And now, GO BACK TO **Weak Map**

weakMap allows Garbage collector to do its Task but Not Map

Map 🡺 Key Can Be Anything

weakMap 🡺 Key Can Be Objs & functins Only

let mapUser = { theName: "Elzero" };

let myMap = new Map();

myMap.set(mapUser, "Obj Value");

mapUser = null; // overide the ref , so there is no ref now

console.log(myMap);

console.log("\*".repeat(22));

// let do the same using weakmap

let wMapUser = { theName: "Elzero" };

let myWeakMap = new WeakMap();

myWeakMap.set(wMapUser, "Obj Value");

wMapUser = null; /// overide the ref , so there is no ref now

console.log(myWeakMap); // as you garbage collection do its task

weakMap 🡺 there is no size property or forEach, iterator methods

**Array.from Method**

The Array.from() method returns an array from any object with a length property .

The Array.from() mehod returns an array from any iterable object.

Syntax :

Array .from( obj, mapFunction, thisValue )

**Parameters**

Obj 🡺 required, the object to convert to an array

mapFunction 🡺 Optional, A map function to call on each item

thisValue 🡺 A value to use As this from the mapFunction

let nameLetters = Array.from("36859", (e) => +e);

console.log(nameLetters); // (5) [3, 6, 8, 5, 9]

**Take some info :**

function testArgs() {

  return arguments;

}

// it returns the arguments entered

console.log(testArgs("Bilal", 698, "nono"));

// Arguments(3) ['Bilal', 698, 'nono', callee: ƒ, Symbol(Symbol.iterator): ƒ]

To make answear clean

function testArgs() {

  return Array.from(arguments);

}

console.log(testArgs("Bilal", 55)); // (2) ['Bilal', 55]

**Array.copyWithin() Method**

The **copyWithin()** method copies array elements to another position in the array .

The **copyWithin()** method overwrites the existing values.

The **copyWithin()** method does not add items to the array.

Syntax

array.copyWithin( Target, Start, End)

**Parameters**

Target 🡺 required, The index (position) to copy the elements to. - if it greater than array length nothing will be copied .

Start 🡺 Optional, the start index (position). Default is 0. - if it’s omitted (neglect) = start from index 0

End 🡺 Optional, the end index (position). Default is the array length . - if it’s omitted = Reach the end (length ) the last index - Not including end

**Return Value**

Type : An array , desc : the changed array

Note :

Any negative value will count from the end.

Array length will not be increased

let myArray = [10, 20, 30, 40, 50, "A", "B"];

// myArray.copyWithin(3);

// (7) [10, 20, 30, 10, 20, 30, 40]

// myArray.copyWithin(1, -2);

// (7) [10, 'A', 'B', 40, 50, 'A', 'B']

myArray.copyWithin(1, -2, -1);

// (7) [10, 'A', 30, 40, 50, 'A', 'B']

console.log(myArray);

**Array.some () Method**

The **some()** method checks if any array elements pass a test (provided as a callback function).

The **some()** method executes the callback function once for each array element .

The **some()** method returns true (and stops) if one array elements passed the test

The **some()** method returns false if no array elements verify the test provided

Syntax

array.some ( function (value, index, arr), this )

**Parameters**

Fucntion 🡺 Required, A function to run for each array element

* **value  :** required, the value of the current element
* **index  :** Optional, the index of the current element
* **arr  :** Optional, The array the current element belongs to.

This 🡺 Optional, Default is undefined, A value passed to the function to be used as its « this » value .

let nums = [1, 2, 3, 4, 5, 6, 7];

let myNumber = 5;

let check = nums.some(function (e, i) {

  console.log(e, i);

  return e > this;

}, myNumber);

console.log(check); // true

**Note :**

this value, works only with simple function not with arrow function.

Example 1 :

function checkValues(arr, val) {

  return arr.some((v) => v === val);

}

console.log(checkValues(nums, 5)); // true

console.log(checkValues(nums, 8)); // false

Example 2  :

function checkNumInRange(arr, min, max) {

  return arr.some((v) => v >= min && v <= max);

}

console.log(checkNumInRange(nums, 5, 10)); // true

console.log(checkNumInRange(nums, 0, 5)); // true

console.log(checkNumInRange(nums, 8, 10)); // false

Example 3  :

let range = {

  min: 5,

  max: 10,

};

let checkNumInRange = nums.some(function (e) {

  return e >= this.min && e <= this.max;

}, range);

console.log(checkNumInRange); // true

let try to change range :

let range = {

  min: 8,

  max: 10,

};

let checkNumInRange = nums.some(function (e) {

  return e >= this.min && e <= this.max;

}, range);

console.log(checkNumInRange); // false

**Array.every () Method**

**every ()** method is like **some ()** method, but there is one important diffrence :

the **every()** method returns true if the function returns true for all elements.

let locations = {

  20: "Place 1",

  30: "Place 2",

  10: "Place 3",

  40: "Place 4",

};

let mainLocation = 5;

let locationsArray = Object.keys(locations).map((e) => +e);

console.log(locationsArray);

let check = locationsArray.every((e) => e >= mainLocation);

console.log(check); // true

**Spread Operator :**

Sperad Operator 🡺 …Iterable

‘allow Iterable to Expand تسع in place’

// Spread With String => Expand String

console.log("Bilal"); // Bilal

console.log(..."Bilal"); // B i l a l

console.log([..."Bilal"]); // (5) ['B', 'i', 'l', 'a', 'l']

// Concatenate Arrays

let myArray1 = [1, 2, 3];

let myArray2 = [4, 5, 6];

console.log(...myArray1, ...myArray2); // 1 2 3 4 5 6

let AllArrays = [...myArray1, ...myArray2];

// (6) [1, 2, 3, 4, 5, 6]

console.log(AllArrays);

// copy Array

let copiedArray = [...myArray1];

console.log(copiedArray); //(3) [1, 2, 3]

// Push inside Array

let friends = ["Bilal", "Ahmad", "Ossama"];

let newFriends = ["Najib", "karim"];

let allFriends = [...friends, ...newFriends];

console.log(allFriends);

//5) ['Bilal', 'Ahmad', 'Ossama', 'Najib', 'karim']

// Use with math Obj

let myNums = [10, 20, 30, 40, 55, 95, 1000, 55];

console.log(Math.max(...myNums)); // 1000

// \*\*\* Spread with Objects ==> Merge Objects

obj1 = {

  a: 1,

  b: 2,

};

Obj2 = {

  c: 3,

  d: 4,

};

console.log({ ...obj1, ...Obj2, e: 5 });

// {a: 1, b: 2, c: 3, d: 4, e: 5}

Notes from Assignment and challenge:

The **entries()** method returns an array iterator with key/value pairs ;

The **entries()** method does not change the original array .

const fruits = ["Banana", "Orange", "Apple", "Mango"];

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

/\*

0: (2) ['0', 'Banana']

1: (2) ['1', 'Orange']

2: (2) ['2', 'Apple']

3: (2) ['3', 'Mango']

\*/

console.log(Object.entries({ name: "Bilal", address: "Bni Makada" }));

/\*

0: (2) ['name', 'Bilal']

1: (2) ['address', 'Bni Makada']

\*/

**Object.assign()** method can convert an object to array :

console.log(Object.assign([], "Bilal"));

// (5) ['B', 'i', 'l', 'a', 'l']

**Regular Expression**

A regular expression is a **pattern** of characters .

The pattern is used to do pattern-mathcing « search-and-replace » function on text.

In Js, a **regExp** Object is a pattern with **properties** and **methods**.

**Syntax :**

/pattern/modidifier(s).

New RegExp(‘’pattern’’, ‘’modifier(s)’’)

**Modifiers 🡺 Flags :**

Modifiers are used to perform case-insesitive and global searches :

Flags :

**g** 🡪 perform a gloabl match (find all matches rather than stopping after the first match)

**i** 🡪 perform case-insesitive matching.

**M** 🡪 perform multiline

// Regular Expression

let myString = "Hello Bilal Dev, I like bilal  !!!";

let regex = /Bilal/;

console.log(myString.match(regex));

console.log(myString.match(/Bilal/));

/// Modifiers

regex = /bilal/i;

console.log(myString.match(regex));

// 'Bilal', index: 6, input: 'Hello Bilal Dev, I like bilal  !!!', groups: undefined]

regex = /bilal/gi; // case insesitive and global match

console.log(myString.match(regex));

// (2) ['Bilal', 'bilal']

// old way :

regex = new RegExp("dev", "ig");

console.log(regex); // /dev/gi

console.log(myString.match(regex));

// ['Dev']

**Ranges :**

**Part 1 :**

(X|Y) 🡪 X or Y

[0-9] 🡪 0 To 9

[^0-9] 🡪 Any character not 0 to 9.

\*\*

// (X|Y)

let tld = "Com Net Org Info Code Io";

let tldRe = /(org|info|io)/i;

console.log(tld.match(tldRe));

// (2) ['Org', 'Org', index: 8, input: 'Com Net Org Info Code Io', groups: undefined]

// By the way we gonna get the same result if we change the arrangement

tldRe = /(info|org|io)/i;

console.log(tld.match(tldRe));

//(2) ['Org', 'Org', index: 8, input: 'Com Net Org Info Code Io', groups: undefined]

// you can change it from Or to And , using g flag

tldRe = /(info|org|io)/gi;

console.log(tld.match(tldRe));

// 3) ['Org', 'Info', 'Io']

\*\*

let nums = "1245679";

let numsRe = /[6-9]/g;

console.log(nums.match(numsRe)); // ['6', '7', '9']

\*\*

let notNumsRe = /[^6-9]/g;

console.log(nums.match(notNumsRe)); // ['1', '2', '4', '5']

**pratice 1 :**

retrun only special caracter :

let specialNums = "1!2@3#4$599\*/10";

let specialNumsRe = /[^0-9]/g;

console.log(specialNums.match(specialNumsRe));

// ['!', '@', '#', '$', '\*', '/']

**pratice 2 :**

let practice = "os1 os10s os2 Os8 Os8Os";

let practiceRe = /Os[5-9]Os/g;

console.log(practice.match(practiceRe)); // ['Os8Os']

**Note :**

The method of matching first, write variable noun, it like entering to the area of work and then use math method and write the pattern with or without flag(s)

**Part 2 :**

[a-z] and [^a-z]

[A-Z] and [^a-z]

[abc] and [^abc] : find any character between bracket

let myString = "AaBbcdefG123!234/^&\*";

let atozSmall = /[a-z]/g;

let notAtozSmall = /[^a-z]/g;

let atozCapital = /[A-Z]/g;

let notAtozCapital = /[^A-Z]/g;

let aAndcAnde = /[ace]/g;

let notaAndcAnde = /[^ace]/g;

let lettersCapsAndSmall = /[a-z]/gi;

// or

let lettersCapsAndSmall2 = /[a-zA-Z]/g;

let numsAndSpecial = /[^a-zA-Z]/g;

console.log(myString.match(atozSmall)); // (6) ['a', 'b', 'c', 'd', 'e', 'f']

console.log(myString.match(notAtozSmall));

// (14) ['A', 'B', 'G', '1', '2', '3', '!', '2', '3', '4', '/', '^', '&', '\*']

console.log(myString.match(atozCapital)); // ['A', 'B', 'G']

console.log(myString.match(notAtozCapital));

// (17) ['a', 'b', 'c', 'd', 'e', 'f', '1', '2', '3', '!', '2', '3', '4', '/', '^', '&', '\*']

console.log(myString.match(aAndcAnde)); // (3) ['a', 'c', 'e']

console.log(myString.match(notaAndcAnde));

// ['A', 'B', 'b', 'd', 'f', 'G', '1', '2', '3', '!', '2', '3', '4', '/', '^', '&', '\*']

console.log(myString.match(lettersCapsAndSmall));

// (9) ['A', 'a', 'B', 'b', 'c', 'd', 'e', 'f', 'G']

console.log(myString.match(lettersCapsAndSmall2));

// (9) ['A', 'a', 'B', 'b', 'c', 'd', 'e', 'f', 'G']

console.log(myString.match(numsAndSpecial));

// (11) ['1', '2', '3', '!', '2', '3', '4', '/', '^', '&', '\*']

**Chracter Classes :**

. 🡪 mathes any character, expect newline or other line terminator ;

\w 🡪 matches word characters : [a-z, A-Z , 0-9 And underscore]

\W 🡪 matches non word Character

\d 🡪 matches digits from 0 t 9

\D 🡪 matches non digit characters.

\s 🡪 matches whitespaces .

\S 🡪 matches non whitespace character.

let email = "O@@@g.cim o@g.com o@g.net A@Y.com o\_g.com o@s.or 1@1.com";

let dot = /./g;

console.log(email.match(dot));

/\* (56) ['O', '@', '@', '@', 'g', '.', 'c', 'i', 'm', ' ', 'o', '@',

'g', '.', 'c', 'o', 'm', ' ', 'o', '@', 'g', '.', 'n', 'e', 't', ' ',

'A', '@', 'Y', '.', 'c', 'o', 'm', ' ', 'o', '-', 'g', '.', 'c', 'o',

'm', ' ', 'o', '@', 's', '.', 'o', 'r', ' ', '1', '@', '1', '.', 'c',

'o', 'm'] \*/

let word = /\w/g;

console.log(email.match(word));

/\*['O', 'g', 'c', 'i', 'm', 'o', 'g', 'c', 'o', 'm', 'o', 'g', 'n',

'e', 't', 'A', 'Y', 'c', 'o', 'm', 'o', '\_', 'g', 'c', 'o', 'm', 'o',

's', 'o', 'r', '1', '1', 'c', 'o', 'm']\*/

let Word = /\W/g; // it the character not included in small w

console.log(email.match(Word));

/\* ['O', 'g', 'c', 'i', 'm', 'o', 'g', 'c', 'o', 'm', 'o', 'g', 'n',

'e', 't', 'A', 'Y', 'c', 'o', 'm', 'o', '\_', 'g', 'c', 'o', 'm', 'o',

's', 'o', 'r', '1', '1', 'c', 'o', 'm']\*/

let valid = /\w@\w.(com|net)/g;

console.log(email.match(valid));

//4) ['o@g.com', 'o@g.net', 'A@Y.com', '1@1.com']

let digit = /\d/g;

console.log(email.match(digit));

// (2) ['1', '1']

let Digit = /\D/g;

console.log(email.match(Digit));

/\* (54) ['O', '@', '@', '@', 'g', '.', 'c', 'i', 'm', ' ', 'o', '@',

'g', '.', 'c', 'o', 'm', ' ', 'o', '@', 'g', '.', 'n', 'e', 't', ' ',

'A', '@', 'Y', '.', 'c', 'o', 'm', ' ', 'o', '-', 'g', '.', 'c', 'o',

'm', ' ', 'o', '@', 's', '.', 'o', 'r', ' ', '@', '.', 'c',

'o', 'm'] \*/

let WhiteSpace = /\s/g;

console.log(email.match(WhiteSpace));

// (6) [' ', ' ', ' ', ' ', ' ', ' ']

let WhiteSpaceCap = /\S/g;

console.log(email.match(WhiteSpaceCap));

/\* (50) ['O', '@', '@', '@', 'g', '.', 'c', 'i', 'm', 'o',

'@', 'g', '.', 'c', 'o', 'm', 'o', '@', 'g', '.', 'n', 'e',

't', 'A', '@', 'Y', '.', 'c', 'o', 'm', 'o', '\_', 'g', '.',

'c', 'o', 'm', 'o', '@', 's', '.', 'o', 'r', '1', '@', '1',

 '.', 'c', 'o', 'm'] \*/

Note :

Small character classes is the opposite of cap character classes , it return character that small didn’t return.

**Part 2 :**

**\b** 🡪 matched at the beginning or end of a word

**\B** 🡪 matches NOT at the beginning or end of a word

If put the class before the word so you mean beginning and if you put after so you mean at the end of a word !!

let names = "Sayed 1Spam 2Spam 3Spam Spam4 Spam5 Osama Ahmed Aspamo";

// spam at the begginning of a word

let re = /\bspam/gi;

console.log(names.match(re)); // (2) ['Spam', 'Spam']

// spam at the end of a word

re = /spam\b/gi;

console.log(names.match(re)); // (3) ['Spam', 'Spam', 'Spam']

\*\*

let text = "Hello, LOOK At you !! ";

let pattern = /\Blo/gi; // sear for "lo" Not at the beginning of a word

console.log(text.match(pattern)); // ['lo']

pattern = /lo\B/gi; // sear for "lo" Not at the end of a word

console.log(text.match(pattern)); // ['LO']

The **test ()** method test for a match in a string .

If it finds a match, it returns true, otherwise it returns false .

let text = "Hello Heros";

// look for "Hero"

let pattern1 = /Hero/gi;

let result1 = pattern1.test(text);

// look for "Weaks"

let pattern2 = /Weaks/gi;

let result2 = pattern1.test(text);

console.log(result1, result2); // true false

console.log(text.match(pattern1)); // Hero

**Quantifiers :**

n+ 🡺 Matches any string that contains at least one n.

let mails = "o@nn.sa osama@gmail.com elezero@gmail.net osama@mail.ru";

let mailsRe = /\w+@\w+.\w+/gi;

console.log(mails.match(mailsRe));

// ['o@nn.sa', 'osama@gmail.com', 'elezero@gmail.net', 'osama@mail.ru']

n\* 🡺 zero or more (from n )

let nums = "0110 10 150 05120 0560 350 00";

let numsRe = /0\d+0/gi;

let numsRe1 = /0\d\*0/gi;

console.log(nums.match(numsRe)); //['0110', '05120', '0560']

console.log(nums.match(numsRe1)); //  ['0110', '05120', '0560', '00']

n ? 🡺 Zero or one (from n )

let urls = "https://google.com http://www.website.new web.com";

let urlsRe = /(https?:\/\/)?(www.)?\w+.\w+/gi;

console.log(urls.match(urlsRe));

// ['https://google.com', 'http://www.website.new', 'web.com']

n{x} 🡺 mathes any string contains X of n

n{X,Y} 🡺 matches any string that contains X to Y of n

n{X,} 🡺 matches any string that contains at least X of n

let serials = "S100S S3000S S50000S S950000S";

// without quantifir n{}

console.log(serials.match(/S\d\d\dS/gi));

// with quantifir n{}

console.log(serials.match(/S\d{3}S/gi)); // S[three numbers]S

// ['S100S']

// to do the next code manually, you will write a lot of code

console.log(serials.match(/S\d{4,5}S/gi)); // S[Four to Five numbers]S

// (2) ['S3000S', 'S50000S']

console.log(serials.match(/S\d{3,}S/gi)); // S[at least 3 numbers]S

//(4) ['S100S', 'S3000S', 'S50000S', 'S950000S']

n$ 🡺 matches any string with n at the end of it .

^n 🡺 Matches any string with n at the beginning of it

?=n 🡺 Matches any string that followed by a specific string n

?!n 🡺 Matches any string that is not followed by a specific string n.

console.log(/ing$/.test(myString)); // true

console.log(/z$/i.test(names)); // true

\*\*

let myString = "We Love Programming";

let names = "1Osamaz 2ahmedZ 3Mohammed 4Moustafaz 5 GamalZ";

console.log(names.match(/\d\w{5}(?=z)/gi));

// (2) ['1Osama', '2ahmed']

console.log(names.match(/\d\w{8}/gi));

// (2) ['3Mohammed', '4Moustafa']

console.log(names.match(/\d\w{8}(?=z)/gi)); // ['4Moustafa']

console.log(names.match(/\d\w{8}(?!z)/gi)); // ['3Mohammed']

**replace & replaceAll**

let text = "We Love programming And @ Because @ is Amazing";

console.log(text.replace("@", "Js"));

// We Love programming And Js Because @ is Amazing

console.log(text.replaceAll("@", "Js"));

// We Love programming And Js Because Js is Amazing

You can use RegEx with replace & replaceAll :

let re = /@/gi;

console.log(text.replace(re, "js"));

// We Love programming And js Because js is Amazing

**Form Validation**

document.getElementById("register").onsubmit = function (e) {

  let phoneInput = document.getElementById("phone").value;

  let phoneRe = /\(\d{4}\)\s\d{3}-\d{4}/gi;

  if (phoneRe.test(phoneInput) !== true) {

    e.preventDefault();

  }

};

**Note :**

You can test your RegEx using some websites :

Serach : Regex js online