**Array Big Introduction**

An Array is a special variable, can hold more than one value .

let webLanguages = ["HTML", "CSS", "JS"];

**why use Arrays ?**

If you have a list of items (a list of all programming languages, for example), storing the cars in single variable is very hard , knowing that there are more than 700 progamming languages .

The solution is an arrray !!

An array can hold many values under a single name , and you can access the values by referring to an index number .

const webLanguages = [

  "HTML",

  "CSS",

  "JS",

  ["jquery", "vue.js", "angular.js", "React.js"],

];

console.log(webLanguages[2]);

and in each index there are indexes , so i can i access to a special character :

console.log(webLanguages[0][2]); 🡺 M

the same thing in arrays :

console.log(webLanguages[3][2][5]); // angular.js[5]   ==>  a

and sure i can edit my array, a changing values :

webLanguages[3] = "Js Frame\_works";

Array Methods

1- length

The length property sets or returns the number of elements is an array .

console.log(webLanguages.length); // 4

and sure using this method we can set the number of elements

webLanguages.length = 5;

but now we have an emty index : webLanguages = emty

so you can increase the number of indexes as you want , and also you can decrease the number of indexes and this will delete some indexes

how to add a index to an array without thouching the others (dynamic:

webLanguages[webLanguages.length] = "new";

how to change the value of the last index  (dynamic) :

webLanguages[webLanguages.length - 1] = "last\_index";

* Unshift :

***The Unshift() method :*** Inserts new elements at the start of an array, and returns the new length of the array.

let cars = ["golf", "Alfa\_Romeo", "BMW"];

cars.unshift("Scoda");

and this method return the new length of the array :

console.log(cars.unshift("Audi")); // 4

Hey you can also insert many elements :

cars.unshift(

  "golf",

  "scoda",

  "dacia",

  " renault",

  "audi",

  "Oppel",

  "Fiat",

  "Porshe"

);

* Push :

***The Push() method :*** Appends new elements to the end of an array, and returns the new length of the array.

And also you can insert many elements .

let cars = ["golf", "Alfa\_Romeo", "BMW"];

cars.push("Buggati", "Ford");

as we said it return the new length of the array :

console.log(cars.push("Rolls-Royce")); // 6

and know we will learn "Removing"

* shift : Removes the first element from an array and returns it.

let cars = ["golf", "Alfa\_Romeo", "BMW"];

cars.shift();

console.log(cars); // ["Alfa\_Romeo", "BMW"];

return the shifted value :

let cars = ["golf", "Alfa\_Romeo", "BMW"];

console.log(cars.shift()); // golf

so you can store it in a variable .

* pop : Removes the last element from an array and returns it.

let cars = ["golf", "Alfa\_Romeo", "BMW"];

cars.pop();

and return it :

first = cars.pop();

console.log(`The Last Element Was ${first}`); // The Last Element Was BMW

sorting Array

1- indexof

The indexOf() method returns the first index (position) of a specified value.

The indexOf() method returns -1 if the value is not found.

The indexOf() method starts at a specified index and searches from left to right.

By default the search starts at the first element and ends at the last.

Negative start values counts from the last element (but still searches from left to right).

let friends = ["Omar", "Saed", "Amin", "Karim", "Omar"];

console.log(friends.indexOf("Omar")); // 0

you can specify the starting point :

console.log(friends.indexOf("Omar", 1)); // 4

and sure negative value for start from the end , but searching is noramally from left to right .

console.log(friends.indexOf("Omar", -3)); // 4

2- lastIndexOf()

Similar to indexof() method , but lastIndexOf() serches the array from the end to the begnning, and also it return -1 if the value is not found .

console.log(friends.lastIndexOf("Omar")); // 4

negative values for specifying the start points :

let friends = ["Omar", "Saed", "Amin", "Karim", "Omar"];

console.log(friends.lastIndexOf("Omar", -1)); // 4

console.log(friends.lastIndexOf("Omar", -2)); // 0

2- include()

**The includes() method returns true if an array contains a specified value.**

**The includes() method returns false if the value is not found.**

**The includes() method is case sensitive.**

Syntax

*array*.includes(*element*,*start*)

let studentPresent = ["Omar", "Saed", "Amin", "Karim", "Omar"];

if (studentPresent.includes("Amin") == true) {

  console.log("Present");

} else {

  console.log("Absent");

}

We can do the same thing with indexof and lastindexof

let studentPresent = ["Omar", "Saed", "Ahamd", "Karim", "Omar"];

if (studentPresent.indexOf("Amin") == -1) {

  console.log("Absent");

} else {

  console.log("Present");

}

3- Sort() :

Sorts the elments of an array alphabetical .

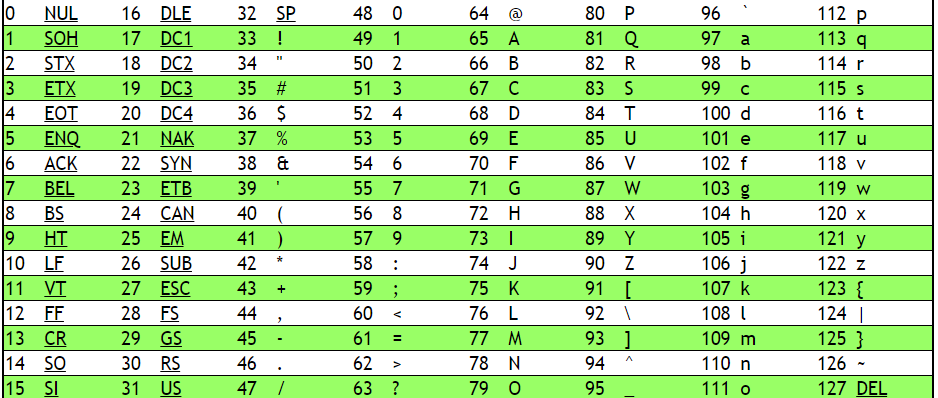
const numbers = ["bilal", "omar", "ahmad", "mariam", "maman", "mas"];

console.log(numbers.sort());

Result : ['ahmad', 'bilal', 'maman', 'mariam', 'mas', 'omar'].

If two words begin by the same letter, Here it compare the second one .

Simply the’re sorted based on ascii table , so numbers always first then uppercase letters then lowercase letters .



const numbers = ["bilal", "omar", "ahmad", "mariam", "maman", "mas","Jake",40,100,50,9];

console.log(numbers.sort());

result : [100, 40, 50, 9, 'Jake', 'ahmad', 'bilal', 'maman', 'mariam', 'mas', 'omar']

4- Reverse() :

The reverse() method reverses the order of the elements in an array.

**Note :** it keep the original order , and just reverse it , the last become the first and vice versa . keep going frinds nofap from grandmother day

const numbers = [

  "bilal",

  "omar",

  "ahmad",

  "mariam",

  "maman",

  "mas",

  "Jake",

  40,

  100,

  50,

  9,

];

console.log(numbers.reverse());

**result :**

**[9, 50, 100, 40, 'Jake', 'mas', 'maman', 'mariam', 'ahmad', 'omar', 'bilal'] .**

And you can get it sorted but reversed :

console.log(numbers.sort().reverse());

result :

**['omar', 'mas', 'mariam', 'maman', 'bilal', 'ahmad', 'Jake', 9, 50, 40, 100]**

5- Slice() : The slice() method returns selected elements in an array, as a new array.

This method take two elment start point included and the end point not included .

**Note : The slice() method does not change the original array.**

Negative value for start counting from the end, but it still working from left two right .

let myFriends = ["Ahmad", "said", "bilal", "omar", "jalil", "monsef"];

console.log(myFriends.slice(2, 4));

console.log(myFriends);

**result :**

['bilal', 'omar'] .

['Ahmad', 'said', 'bilal', 'omar', 'jalil', 'monsef'].

As you see this method doesn’t touch the origin array, but it return new array , and keep the real one as it is .

So you can save the new array in a variable , and as we said negative for counting from the end :

let schoolFriends = myFriends.slice(-4, -2);

you can work with positive and negative at the same time .

console.log(myFriends.slice(1, -1));

6- Splice() :

The splice() method add and/or removes array elements

Syntax

*array*.splice(*index*,*howmany*,*item1*, .....,*itemX*)

**index : Required**

**the position to add/remove items**

**Negative value defines the position from the end of the array .**

**Howmany : optional**

**Number of items to removed**

**Item1 …. , itemx : optional**

**New element(s) to be addes.**

let myFriends = ["Ahmad", "said", "bilal", "omar", "jalil", "monsef"];

myFriends.splice(-2, 2, "yousef", "azdin");

Here we delete 2 elments from the second item from the end of the array , and then we added two indexes in array , in second index form the last of array .

Result **: ['Ahmad', 'said', 'bilal', 'omar', 'yousef', 'azdin']**

7- concat() :

**The concat() method concatenates (joins) two or more arrays.**

**The concat() method returns a new array, containing the joined arrays.**

**The concat() method does not change the existing arrays.**

Syntax

*array1*.concat(*array2*,*array3*, ...,*arrayX*)

et cheapCars = ["Dacia", "Renault", "Fiat"];

let expensiveCars = ["porshe", "BMW", "Ferrarai"];

let mediumCars = ["scoda", "opel", "Citroën"];

console.log(cheapCars.concat(expensiveCars, mediumCars, cheapCars));

Result :

**['Dacia', 'Renault', 'Fiat', 'porshe', 'BMW', 'Ferrarai', 'scoda', 'opel', 'Citroën', 'Dacia', 'Renault', 'Fiat'] .**

You can add a string also or a stirng in concat methods .

 cheapCars.concat(expensiveCars, mediumCars, cheapCars, "golf", "clio", [

    "zibra",

    "honda",

    "suzuki",

    "rangRover",

  ])

And sure this mothod create new array so this method does not change the existing arrays .

8- join() :

**The join() method returns an array as a string.**

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

**Any separator can be specified. The default is comma (,).**

Syntax

*array*.join(*separator*)

let allCars = cheapCars.concat(expensiveCars, mediumCars);

console.log(allCars.join());

**Result : Dacia,Renault,Fiat,porshe,BMW,Ferrarai,scoda,opel,Citroën**

***Separator*: Optional.**

**The separator to be used , defulat is a comma .**

**So Here I will make space as a sperator which separat allcars array items :**

console.log(allCars.join(" "));

**Result : Dacia Renault Fiat porshe BMW Ferrarai scoda opel Citroën**

Note : this return return string .

let nums = [15, 5, 58, 68, 58, 58];

console.log(nums.join(" "));

Result : 15 5 58 68 58 58 (as a string)

So you can use stirng methods .

Loop

Loops can execute a block of code a number of times .

Javascript supports different kinds of loops .

**The For Loop .**

The for statement creates a loop with 3 optional expression :

**for (*expression 1*;*expression 2*;*expression 3*) {  
  // *code block to be executed*  
}**

**expression 1** is executed (one time) before the execution of the code block.

**expression 2** defines the condition for excuting the code block

**expression 3** is executed (every time) after the code block has been executed .

imagine with me, that your boss told you that you campony need 10 employee, and now you have to bring these 10 employee,each time you find one you give him the result , after each one, one is miss .

for (let i = 0; i < 10; i++) {

  console.log(`Now we need ${10 - i} employee`);

}

console.log("Sorry, we don't need any employee now");

**expression 1 :** sets a variable before the loop starts (let i = 0 )

**expression 2 :** defines the condition for the loop to run (i must be less than 10 )

**expression 3 :** increases the value each time by one.

**Expression 1**

Normally you will use expression 1 to initialize the variable used in the loop .

This is not always the case, js doesn’t care , Expression 1 is optional .

You can initiate many values in expression 1 (separated by comma) :

const names = [

  "Bilal",

  "testing",

  "ossama",

  "khaled",

  "ahmad",

  "marwan",

  "reda",

  "saed",

  "eman",

  "alex",

];

for (let i = 0, len = names.length, text = " "; len > i; i++) {

  text += `${names[i]} \n  \n`;

  if (i === len - 1) {

    console.log(

      `%c${text}`,

      "color:dodgerblue;font-size:20px;font-family:cursive"

    );

  }

}

Why we make if condition , because text is created in the loop so we can’t access to it outside the loop,and for make sure that we will log all items in the console , we print it , after make sure that all indexes added to index .

Note : and you can omit the first expression (like when you values are set before the loop start :

let y = 0;

for (; y < 5; y++) {

  console.log(`try number ${y}`);

}

**Expression 2**

Often expression 2 is used to evaluate the condition of the initial variable .

This is not always the case, js doesn’t care. Expression 2 is also optional .

If expression 2 returns true, the loop will start over again, if it returns false, the loop will end .

Note : if you omit expression 2 , you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash you browser. In the next chapters we’ll learn a lot about break .

let i = 0;

for (;;) {

  console.log(i);

  if (i == 5) {

    break;

  }

  i++;

}

**Expression 3**

Often expression 3 increments the value of the initial variable ;

But that not always , expression 3 is optional.

Expression 3 can do anything like decreament (i--), positive increment (i=i+15),or anything else.

Expressin 3 can be omitted (like when you increment you values inside the loop

let i = 0;

for (; i < 5; ) {

  console.log(i);

  i++;

}

**Nested loop :**

let products = ["keyboard", "Mouse", "Pen", "Pad", "Monitor", "Microphone"];

let colors = ["Black", "white", "DarkBlue", "Gray"];

let models = [2020, 2021, 2022];

console.log("%cOur Products \n", "font-size:40px;color:hotpink; ");

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

  console.log("#".repeat(50));

  console.log("#".repeat(50));

  console.log(

    `%c${products[i]}`,

    "font-size:30px;color:olive;font-weight:bold"

  );

  console.log("%c  Colors : ", "font-size:20px;color:greenyellow");

  for (let y = 0; y < colors.length; y++) {

    console.log(`                 - ${colors[y]}`);

  }

  console.log("%c  Models : ", "font-size:20px;color:dodgerblue");

  for (let x = 0; x < models.length; x++) {

    console.log(`                  - ${models[x]}`);

  }

}

We will do like a calender :

for (let i = 0; i < 4; i++) {

  console.log(`Week : ${i + 1}`);

  for (let y = 0; y < 7; y++) {

    console.log(`     Day : ${y + 1}`);

  }

}

**Loop control : Break, continue, Label :**

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

## The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

The break statement can also be used to jump out of a loop:

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

  if (i === 6) {

    break;

  }

  console.log(i + 1);

}

IN the example above, the break statement ends the loop (« breaks » the loop), when the loop counter reaches number : ‘6’

## The Continue Statement

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips all number values and stops when it reches Stop index:

const list = [

  "Bilal",

  "Ahmad",

  "Omar",

  15,

  "Hamid",

  20,

  52,

  "Yasmin",

  "Stop",

  "Alex",

];

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

  if (list[i] === "Stop") {

    break;

  }

  if (typeof list[i] === "number") {

    continue;

  }

  console.log(list[i]);

}

## JavaScript Labels

To label JavaScript statements you precede the statements with a label name and a colon:

let fruits = ["Banana", "Apple", "peach", "apricot"];

let colors = ["Green", "Yellow", "Black", "Orange"];

mainLoop: for (let i = 0; i < fruits.length; i++) {

  console.log(fruits[i]);

  nestedLoop: for (let y = 0; y < colors.length; y++) {

    if (colors[y] === "Black") {

      console.log("  !! There is a problem in this fruits ");

      continue mainLoop;

    } else {

      console.log(`      - ${colors[y]}`);

    }

  }

}

Here in this example we stop mainLoop using label if we find a black fruits and log in console a message and tell us about the problem .

Adding Product to the page :

const products = [

  "Keyboard",

  "Mouse",

  "Pen",

  "Pad",

  "Pc Gamer",

  "Heardset",

  "WebCam",

  "Monitor",

];

const Colors = ["Black", "White"];

const Models = [2021, 2022];

const prices = [20, 15, 10, 7, 900, 20, 30, 250];

document.write("<h1>Our Products </h1>");

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

  document.write(

    `<div class="container" Style="display:flex; justify-content:center ; gap : 30%;flex-wrap:wrap">`

  );

  // left side

  document.write(`<div >`);

  document.write(

    `<h2 style="color:olive;font-size:30px;font-family:cursive">${products[i]}</h2>`

  );

  document.write(

    `<p><span style="font-weight: bold;font-size:22px">Colors : </span>${Colors.join(

      " | "

    )}</p>`

  );

  document.write(

    `<p><span style="font-weight: bold ;font-size:22px">Models : </span>${Models.join(

      " | "

    )}</p>`

  );

  document.write(

    `<span style="font-weight: bold ; font-size:25px">Prices : $${prices[i]}</span>`

  );

  document.write(` </div>`);

  // right side

  document.write(`<img src="download.jfif" />`);

  // finish

  document.write(`  </div> <hr >`);

}

document.querySelector("body").style.cssText =

  "font-family:sans-serif;font-size:20px";

document.querySelector("h1").style.cssText =

  "font-size:50px ; text-align:center ; color:olive";

Javascript While Loop

Loops can execute a block of code as long as a specified condition is true.

The while loop loops through a block of code as long as a specified condition is true.

### **Syntax**

while (condition) {  
*// code block to be executed*  
}

let i = 0;

while (i < 10) {

  console.log(i + 1);

  i++;

}

Note :

If you forget to increase the variable used in the condition , the loop will never ends . this will crassh you browser

Javascript Do While Loop

The do while loop is a variant of the while loop . this loop will execute the code block once , before checking if the condition is true , then it will repeat the loop as long as the condition is true .

### **Syntax**

do {  
*// code block to be executed*}  
while (condition);

do {

  console.log("good");

} while (false);

It will log « good » in the console, although the condition false , but it will stop after checking the condition .

**Function**

A JavaScript fuction is a block of code designed to perform a particular task .

A javaScript function is excuted when « something » invokes it (calls it ) .

**User Defined function vs Built In function**

User defined function is a function that you made by yourself .

Built IN fuction is a language function (ready to use) .

**JavaScript Function Syntax :**

A javaScript function is defined with function keyword,followed by a **name**, followed by parentheses () .

Function names can contain letters,digits,underscores,and dollar signs (same rules as variable)

The parentheses may include parameter names separated by commas :

(**parameter1** , **parameter2**, …..)

The code to be excuted, by the function is placed inside curly brackets { }

function sayHello(userName) {

  console.log(`Hello ${userName}`);

}

Function **parameters** are listed inside the parantheses () in the function definition .

Function **arguments** are the values received by the function when it is invoked .

**Inside the function, the arguments (the parameters) behave as local variables.**

Another example :

function driveLicense(userName, age) {

  console.log(`Hello ${userName} Your age is ${age}`);

  if (age >= 18) {

    console.log(`Good, you can get a license drive`);

  } else if (age < 18) {

    console.log(

      `Sorry you are too young to get a license drive now, please wait ${

        18 - age

      } year(s) `

    );

  }

}

A very good example :

function generateYears(start, end) {

  for (let y = 0; start <= end; start++, y++) {

    console.log(`${start}  ==> ${y + 1}`);

  }

}

generateYears(1979, 2022); // my mom life

i can exclude a year :

function generateYears(start, end, exclude) {

  for (; start <= end; start++) {

    if (start === exclude) {

      continue;

    }

    console.log(start);

  }

}

**Function Return :**

When Javascript reaches a return statement, the function will stop excuting .

Function often compute a return value. The return value is ‘returned’ back to the ‘caller’ .

function calc(num1, num2) {

  return num1 \* num2;

}

console.log(calc(9, 6)); // 54

and sure i can store the returned value in a variable .

function calc(num1, num2) {

  return num1 \* num2;

}

let result = calc(9, 6);

console.log(result + 10); // 64

Note : you can write return in a line and returned value in onther line , Vs code will automatically add a semicolon and sure returned value will be unrechable code .

Take this :

function genarate(start, end) {

  for (; start <= end; start++) {

    if (start === 15) {

      return 'interruptin : "cutting"';

    }

    console.log(start);

  }

}

**Function Default Parameters**

Imagine with me that you create a sayHello() functon but the user doesn’t enter his name :

function sayHello(userName) {

  return `Hello Mr(s) ${userName}`;

}

console.log(sayHello()); // Hello Mr(s) undefined

But me i don’t want to print undefined if no name, because the user will not understand that. But i want to print a particular text if there is no userName .

**1- first solution :**

function sayHello(userName) {

  if (userName == undefined) {

    userName = "(Unkown)";

  }

  return `Hello Mr(s) ${userName}.`;

}

console.log(sayHello()); // Hello Mr(s) (Unkown).

**2-Truthy and falsy**

In JavaScript a truthy value that is considered **true** when put it in a **Boolean context**. All values are truthy unless they are defined as falsy. That is, all values are truthy **except false**, 0, -0, 0n, "", null, undefined, and NaN.

function sayHello(userName, age) {

  if (Boolean(age) === false) {

    age = "Unkonw";

  }

  return `Hello ${userName} Your Age is ${age}`;

}

But there a good way to do this :

function sayHello(userName, age) {

  age = age || "Unkonw";

  return `Hello ${userName} Your Age is ${age}`;

}

This trick check if age a falsy value , it is , so age will equal « Unkonw »

**3- check for nulll and undefined only**

As the last one but this doesn’t check for all falsy values , it check only for null and undefined . replace double « Pipes «  by double question mark .

function sayHello(userName, age) {

  age = age ?? "Unkonw";

  return `Hello ${userName} Your Age is ${age}`;

}

**4- ES6 Method (modern and new method)**

function sayHello(userName = "Unkown", age = "Unkown") {

  return `Hello ${userName} Your Age is ${age}`;

}

**Rest Parameters**

Now you have to make a calculator that numbers that i give them to it .

function addCalc(num1 = 0, num2 = 0, num3 = 0) {

  return num1 + num2 + num3;

}

console.log(addCalc(10, 35, 96)); /// 141

console.log(addCalc(10, 35)); /// 45

Now imagine with me, that i give the function more than tree numbers , i am gonna give 10 or 20 numbers, and sure you can’t still creating parameters, but you have to use a dynamique trick . that is **Rest Parameter**

The **rest parameter** syntax allows a function to accept an indefinite number of arguments as an array.

unction addCalc(...num) {

  console.log(Array.isArray(num));

}

addCalc(); // true

so rest parameter accept unlimite of argument and store them in array ‘…num ‘

and know for print the sum of this numbers :

function addCalc(...numbers) {

  let result = 0;

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

    result += numbers[i];

  }

  return result;

}

console.log(addCalc(10, 2, 10, 358, 58)); // 438

**Ultimate Function Practice :**

function showInfo(

  userName = "Unkown",

  userAge = "Unkown",

  userRate = "0",

  showSkills = "yes",

  ...userSkills

) {

  document.write(`<div>`);

  document.write(`<h2>Welcome, ${userName}</h2>`);

  document.write(`<p>Age: ${userAge}</p>`);

  document.write(`<p>Hour Rate:  $${userRate}</p>`);

  if (showSkills === "yes") {

    if (userSkills.length > 0) {

      document.write(`<p> Skills : ${userSkills.join(" | ")}</p>  `);

    } else {

      document.write(`<p> Skills : There is No Skills</p>  `);

    }

  } else {

    document.write(`<p>Skills are Hidden  !!</p>`);

  }

  document.write(` <hr> </div>`);

}

showInfo(

  "Bilal Elemrani",

  17,

  30,

  "yes",

  " Html ",

  " Css ",

  " Js ",

  " Bootstrap",

  "node.Js"

);

showInfo(

  "Omar Abo Kassim",

  19,

  35,

  "yes",

  "Adobe Photoshop",

  "Adobe Premier",

  "Cinema 4D "

);

showInfo("Mohammed Elemrani", 41, 20, "yes", "designing", "Cutting", "Sewing ");

showInfo("Ahmad Slimani", 41, 20, "yes");

showInfo("Karim Moulodi", 17, 30, "no", "voice ff");

**Challenge**

My first try :

You can put the name in first parameter or the second one , and age too, but c is reserved for boolean value .

function userInfo(a = "Unkonw", b = "Unkonw", c = true) {

  let availibity = c === true ? "Avialible" : "Not Avialble";

  if (typeof a === "string") {

    return `Hello ${a}, your age is ${b}, you are ${availibity} For Hire`;

  } else {

    return `Hello ${b}, your age is ${a}, you are ${availibity} For Hire`;

  }

}

console.log(userInfo("Bilal", 20, false));

and now i’ll do the same thing but with ternary operator :

function userInfo(a = "Unkonw", b = "Unkonw", c = true) {

  let availibity = c === true ? "Avialible" : "Not Avialble";

  return typeof a === "string"

    ? `your name is ${a} and your age is ${b}, your are ${availibity} For Hire`

    : `your name is ${b} and your age is ${a}, your are ${availibity} For Hire`;

}

console.log(userInfo("Bilal", 20, false));

but know i made it !!! but using if statemnt and some teranry operator but i don’t use ternary operator only :

function userInfo(a = "Unkonw", b = "Unkonw", c = "Unkown") {

  if (typeof a === "string") {

    if (typeof b === "number") {

      return `Hello ${a}, your age is ${b}, your are ${(c =

        c === true ? `avialable` : `not availible`)} For Hire`;

    } else {

      return `Hello ${a}, your age is ${c}, your are ${(b =

        b === true ? `avialable` : `not availible`)} For Hire`;

    }

  } else if (typeof b === "string") {

    if (typeof a === "number") {

      return `Hello ${b}, your age is ${a}, your are ${(c =

        c === true ? `avialable` : `not availible`)} For Hire`;

    } else {

      return `Hello ${b}, your age is ${c}, your are ${(a =

        a === true ? `avialable` : `not availible`)} For Hire`;

    }

  } else {

    if (typeof b === "number") {

      return `Hello ${c}, your age is ${b}, your are ${(a =

        a === true ? `avialable` : `not availible`)} For Hire`;

    } else {

      return `Hello ${c}, your age is ${a}, your are ${(b =

        b === true ? `avialable` : `not availible`)} For Hire`;

    }

  }

}

console.log(userInfo(17, true, "Bilal"));

ok, i find the good answear :

function userInfo(a = "Unkonw", b = "Unkonw", c = "Unkown") {

  let name;

  let age;

  let status;

  name = typeof a === "string" ? a : typeof b === "string" ? b : c;

  age = typeof a === "number" ? a : typeof b === "number" ? b : c;

  status = typeof a === "boolean" ? a : typeof b === "boolean" ? b : c;

  return `Hello ${name}, Your Age Is ${age}, You Are ${(status =

    status === true ? "Available" : "Not Available")} For Hire`;

}

console.log(userInfo(15, true, "bilal"));

i made it, by onther way :

function userInfo(a = "Unkonw", b = "Unkonw", c = "Unkown") {

  let name;

  let age;

  let status;

  typeof a === "string"

    ? (name = a)

    : typeof b === "string"

    ? (name = b)

    : (name = c);

  typeof a === "number"

    ? (age = a)

    : typeof b === "number"

    ? (age = b)

    : (age = c);

  typeof a === "boolean"

    ? (status = a)

    : typeof b === "boolean"

    ? (status = b)

    : (status = c);

  return `Hello ${name}, Your Age Is ${age}, You Are ${(status =

    status === true ? "Available" : "Not Available")} For Hire`;

}

**Anonymous Function**

**Anonymous Function** is a function that does not have any name associated with it.

An anonymous function is not accessible after its initial creation

 it can only be accessed by a variable it is stored in as a *function as a value.*

let calculate = function (number1, number2) {

  return number1 + number2;

};

calculate(5, 5);

normal function is accesible after its initialization :

calc(20, 5);

function calc(num1, num2) {

  return(num1 + num2);

}

But anonymous function can be accessible neither before nor after its initialization,but if you store the anonymous function in a variable you will be able to access after its initialization using variable name , but as variable you can’t access it before initialization, the function the same :

console.log(name); // Error

let name = "Bilal";

the function the same :

calculate(5, 5); // Error

let calculate = function (number1, number2) {

  return number1 + number2;

};

So anonymous function created in runtime , so you can’t access it before .

**But why we use anonymous function :**

We use anonymous function in :

1- function that we use it one time , by example in a button .

document.getElementById("show").onclick = function () {

  console.log("show");

};

But also you can use a function that is all ready existe :

document.getElementById("show").onclick = sayHello;

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

1 second = 1000 milliseconds.

setTimeout(function () {

  console.log("Hello");

}, 1000);

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

**If you need repeated executions, use setInterval() (الفاصل الزمني ) instead.**

setInterval(function () {

  console.log("Hello");

}, 1000);

**Nested Function**

We use nested function simplify and to reduce the difficulty of a task, so intead of using one function in one tas, you Divide the problem to many small problem that are easy to be solved .

Try to understand this code, this is a challenge for you :

mainFunction: function sendMessage(firstName, lastName) {

  let sms = "Hey";

  // nested Function

  first\_Nested\_Function: function concatMsg() {

    // Nested Function

    second\_Nested\_Function: function concatName() {

      return `${firstName} ${lastName}`;

    }

    return `${sms} ${concatName()}`;

  }

  return concatMsg();

}

console.log(sendMessage("Bilal", "Elemrani"));

**Arrow Function**

Arrow Function were introduced in ES

Arrow Function allow us to write shorter function syntax :

**Before :**

function sayHello() {

  return `Hello Everyone`;

}

**After :**

let myFunction = let myFunction = () => {

return "Hello World!";

};

It gets shorter ! if the function has only one statements, and the statement returns a value, you can remove the brackets and the **return** Keyword

let myFunction = () => `Hello Everyone`;

let myFunction = (a, b) => a \* a + b;

in Fact, if you have only one parameter, you can skip the parentheses as well :

let myFunction = a => a \* a + 2;

and if you have no parameters you can skip parentheses too, and replace them by unserscore :

let myFunction = \_ => 6;

**JavaScript Scope**

Scope determines the accessibility (visibilty) of variables .

Javascript has 3 types of scope :

* Block scope
* Function scope (local scope)
* Global scope

1- Global and Local Scope

Variables Declared Globally (outside any function) have Global Scope .

Global Variables can be accessed from anywhere in a javascript program.

Javescript has function scope or local scope : each function create a new scope .

Variables defined inside a function are not accessible (visible) from outside the function.

var a = 1;

let b = 2;

function sayHello() {

  console.log(`Function == From Global ${a}`);

  console.log(`Function == From Global ${b}`);

}

sayHello();

console.log(`From Global ${a}`);

console.log(`From Global ${b}`);

as you are seeing, variables declared outside a function has global scope, this why i could access them inside a function .

and as we said each function has its scope, so when you use a variable the function start looking for it inside its scope first, and if it doesn’t find the variable it starts looking for in global scope :

var a = 1;

let b = 2;

function sayHello() {

  var a = 10;

  let b = 20;

  console.log(`Function == From Local ${a}`);

  console.log(`Function == From Local ${b}`);

}

sayHello();

console.log(`From Global ${a}`);

console.log(`From Global ${b}`);

**result :**

Function == From Local 10

Function == From Local 20

From Global 1

From Global 2

As we saw the function use the existed variables in her scope instead of using glbal ones .

function sayHello() {

  console.log(`Function == From Global ${a}`);

  console.log(`Function == From Global ${b}`);

  var a = 10;

  let b = 20;

}

Here we will get an error, because the function has variables in its scope, but i try to access them before their initiallization

**Make in mind that you can’t access a variable witch created inside a function, so you can’t pass from global to local , but you can pass from local to global, try to understand**

2- Block Scope :

Before Es6, JavaScript had only **Global Scope** and **Function Scope.**

**Block scope** is made espicilaly for **block of code** like : if , switch , for , while…

Now, i’ll give a Qu, what the result of this :

var x = 10;

if (true) {

  var x = 20;

}

console.log(x);

result : 20

but this break the law that we learnt (local scope and global scope)

the cause is **Var** keyword can NOT have Block scope.

ES6 introduced two important new JavaScript keywords: let and const.

These two keywords provide **Block Scope** in JavaScript.

Variables declared inside a { } block can be accessed from outside the block.

{

    var x = 2;

  }

  // x CAN be used here if you use var in {}

So here we will have a problem that the code inside {} overide the code and ruin all the program, so for avoding all this DON’T USE VAR KEYWORD

So for follow the law and make a new scope for each block of code ,

Use new ES6 keyword let , const

**Scope lexical**

All the idea here is that you can pass from scope of child function to scope of parent function .

**Composite Functions**

let calc = (x) => {

  let b = Math.sqrt(x); // the first function

  let Calc = () => {

    return b / (1 + b); the second function

  };

  return Calc();

};

console.log(calc(9));

and :

let calc = (x) => {

  let b = x + 1;

  let Calc = () => {

    return b \* b;

  };

  return Calc();

};

You store the first in a variable,the you use it in the second function .

**Currying Function Technique**

Curying is an advanced technique of working with function.it’s used not only in JavaScript. But in other languages as well.

Currying as a transformation of functions that translates a function from callable as f(a,b,c) into callable as f(a)(b)(c) .

let message = (name) => (time) => (age) => {

  return `${(time = "morning"

    ? "Good Morning"

    : "Good Evening")} Bilal, Your Age is ${age}`;

};

console.log(message("Bilal")("morning")(15));

instead of :

let message = (name, time, age) => {

  return `${

    time == "morning" ? "Good Morning" : "Good Evening"

  } ${name}, Your Age is ${age}`;

};

console.log(message("Bilal", "evening", 17));

take a look :

function great(greeting) {

  return (name) => {

    return greeting + `, ` + name + " !!";

  };

}

let morningGreeting = great("Good morning");

let nightGreeting = great("Good night");

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

console.log(nightGreeting("dad"));

**Higher Order Function**

**Definition :** higher order function is a function that receives a function as an argument and / or returns the function as output.

**1- map :** is an array method

\*\*\*\*\*\*\*\*\*\*\*\*Create new array

Higher order functions loop on every element in an array.

Example :

let nums = [1, 2, 3, 4, 5, 6, 7, 8, 9];

let newArray = [];

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

  newArray.push(nums[i] \* 2);

}

Let try to do this, using map :

Ooooo, where is syntax ????

***array*.map(function(currentValue, index, arr), thisValue)**

**function () : required ;** a function to be run for each array element

**currentValue : required** the value for the current element

index : optional The index of the current element

arr : Optional the array for the current element

thisValue Optional dafault value is undefined, A value passed to the function to be used as its this value .

see this code will help you to understand more and more ….

let newArray = nums.map(function (currentValue, Index, Array) {

  console.log(`currentValue  ==== > ${currentValue}`);

  console.log(`Index  ==== > ${Index}`);

  console.log(`This Argument (value)  ==== > ${this}`);

  console.log(`Array  ==== > ${Array.join(" ")}`);

  console.log(`\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*`);

}, 2);

**Note :** you can name variables with any name that you want it.

**And Now let make it uisng map**

newArray = nums.map(function (ele) {

  return ele \* 2;

});

Or :

newArray = nums.map((currentValue) => currentValue \* 2);

or you can use onther function :

let addition = (x) => x \* 2;

newArray = nums.map(addition);

as we know the first element for currentValue the second one for index ….

map() does not change the original array.

**Practice 1 :**

let swappingCases = "elZERo";

let swap = swappingCases

  .split("")

  .map((el) => (el.toLowerCase() === el ? el.toUpperCase() : el.toLowerCase()))

  .join("");

console.log(swap);

**Practice 2 :**

let invertNumbers = [1, -10, -20, 15, 100, -30];

let inv = invertNumbers.map((el) => -el);

console.log(inv);

**Practice 3 :**

let ignoreNumbers = "Elz123er4o";

let ign = ignoreNumbers

  .split("")

  .map((el) => (isNaN(el) ? el : ""))

  .join("");

console.log(ign);

**2- Filter :** is an array method

\*\*\*\*\*\*\*\*\*\*\*\*Create new array

Filter has the same Syntax with map but it retruns only elements that pass a test provided by a function

let friends = ["Ahmed", "Samed", "Sayed", "Asmaa", "Amgad", "Israa"];

let filteredFriends = friends.filter((element) => element.startsWith("A"));

so filter will return only elments that start with « A » in other word only the element that verify the condition and the condition returns true in it .

// get Even Numbers Only

let numbers = [11, 20, 2, 5, 17, 10];

let evenNumbers = numbers.filter((el) => el % 2 == 0);

let oodNumbers = numbers.filter((el) => el % 2 !== 0);

***Practice 1 :***

let sentence = "I Love Foood Code Too Playing Much";

let filterdSentence = sentence

  .split(" ")

  .filter((el) => el.length <= 4)

  .join(" ");

***Practice 2 :***

let ignoreNumbers = "Elz123er4o";

let ign = ignoreNumbers

  .split("")

  .filter((el) => isNaN(parseInt(el)))

  .join("");

***Practice 3 :***

// Filter String + Multiply

let mix = "A13BS2ZX";

let solution = mix

  .split("")

  .filter((el) => !isNaN(parseInt(el)))

  .map((el) => el \* el);

**3- Reduce**

\*\*\*\*\*\*\*\*\*\*\*\*Create new array

Syntax :

***array*.reduce(function(total, currentValue, currentIndex, arr), initialValue).**

**Parameters :**

\* Reducer function parameters

|  |  |
| --- | --- |
| **total** | **Required**  The initialValue, or the previoulsy returned value of the function |
| **currentValue** | **Required**  The value of the current element |
| **currentIndex** | **Optional**  The index of the current element |
| **arr** | **Optional**  The array the current element belongs to |

|  |  |
| --- | --- |
| **initialValue** | **Optional.**  A value to be passed to the function as the initial value . |

**Return Value :**

The accumulated result from the last call of the callback function.

let nums = [10, 20, 15, 30];

let addition = nums.reduce(function (

  accumulatedValue,

  currentValue,

  currentIndex,

  array

) {

  console.log(`previousValue ==> ${accumulatedValue}`);

  console.log(`currentValue  ==> ${currentValue}`);

  console.log(`index         ==> ${currentIndex}`);

  console.log(`array         ==> ${array}`);

  console.log(accumulatedValue + currentValue);

  console.log(`\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*`);

  return accumulatedValue + currentValue;

});

Total or accumulator or previousValue or accumulatedValue is the value returned from the previous operation .

The question is what is total in index 0 :

*If there is an InitialValue so it is total (function starts from index 0 )*

*If there is no intialValue so the total is value in index 0 (function starts from index 1 )*

***Practice 1 :***

let theBiggest = ["Bla", "Propaganda", "Other", "AAA", "Battery", "Test"];

let theBiggestWord = theBiggest.reduce((acc, cu) =>

  acc.length <= cu.length ? cu : acc

);

***Practice 2 :***

let removeChars = ["E", "@", "@", "L", "Z", "@", "@", "E", "R", "@", "O"];

let rightWord = removeChars.filter((cu) => cu !== "@").join("");

let replace join with « reduce «

let removeChars = ["E", "@", "@", "L", "Z", "@", "@", "E", "R", "@", "O"];

let rightWord = removeChars

  .filter((cu) => cu !== "@")

  .reduce((acc, cu) => `${acc}${cu}`);

**4- forEach**

Method excutes a provided function once for each array element .

*array*.forEach(function(currentValue, index, arr), thisValue).

**Note :**

- doesnt Return Anything [Undefined] .

- Break Will Break The Loop

**Example :**

In a folder behind this file .

**Object**

IN real life, a car is an object .

A car has **properties** like weight and color, and **methods** like start and stop :

All cars have the **same properties,** but the **property values differ from car to car .**

All cars have the **same mehtods**, but the **methods are performed at different times** .

**JavaScript Objects**

let user = {

  // properties

  theName: "Bilal",

  theAge: 16,

  // Methods

  sayHello: () => `Hello`,

};

**Accessing Object Properties :**

***objectName.propertyName***

console.log(user.sayHello(), user.theName);

but we have onother way .

# **Dot Notation vs Bracket Notation**

In the object.propertyName syntax, the propertyName must be a valid js identifier.

But if the propertyName is a valid js identifier so there is no diffrents between using Dot Notation and using Bracket Notation.

In not valid identifier you should use bracket Notation.

let user = {

  // properties

  theName: "Bilal",

  theAge: 16,

  "country Of": "Morocco",

  // Methods

  sayHello: () => `Hello`,

};

// accessing using Bracket Notation

console.log(user["country Of"]);

and as we say in valid identifier use what you want :

console.log(user.theName);

console.log(user["theName"]);

see this :

let myVar = "country";

let user = {

  // properties

  theName: "Bilal",

  country: "Germany",

};

console.log(user[myVar]); // Germany

**Nested Object :**

This is an example about nested object :

let user = {

  theName: "Bilal",

  theAge: 16,

  skils: ["HTML", "CSS", "Js", "Python"],

  availibility: true,

  addresses: {

    inTangier: {

      main: "Bni Makada",

      second: "Mesnana",

    },

    inRabat: "Bwitat",

  },

  checkAv: () =>

    user.availibility === false

      ? `The User is not Available`

      : "The User is Available",

};

**1- accessing to an array :**

// Accessing By Index

console.log(user.skils[user.skils.indexOf("Js")]);

-------or

console.log(user.skils[3]);

or you can show all array element :

console.log(user.skils.join(" | "));

**2-accessing to a nested object :**

console.log(user.addresses.inTangier.main);

**Create Object With New Keyword:**

***1- Accessing To Object Data :***

You can add or modify any object property :

let user = { name: "Ahmad" };

console.log(user.name); // Ahmad

(user.name = "Bilal"), (user.availability = true);

console.log(user.name); // Bilal

as you see i change the value of ‘name’ property and I add another property name ‘availability’

***1- Create Object With New Keyword :***

let User = {};

let user = new Object({});

these two lines of code they are the same .

**second line :** create an object , and we know in js , almost ‘everything’ is an object .

* Booleans can be objects (if defined with the new keyword)
* Numbers can be objects (if defined with the new keyword)
* Strings can be objects (if defined with the new keyword)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects (wait for it)
* Arrays are always objects
* Functions are always objects
* Objects are always objects

So :

let user = new Object([]); // I have created an Array

and you can input data in it directly and indirectly

let user = new Object([15, 5]); // to input directly and indirectly

console.log(user); // (2) [15, 5]

user.push(4, 6); // to input indirectly

console.log(user); // (4) [15, 5, 4, 5]

- the same with object object :

let user = new Object({ name: "Bilal", number: 19, class: "1Bac SMF" }); // to input directly and indirectly

console.log(user); // {name: 'Bilal', number: 19, class: '1Bac SMF'}

user.lang = "HTML CSS JS"; // to input indirectly

console.log(user); // {name: 'Bilal', number: 19, class: '1Bac SMF', lang: 'HTML CSS JS'}

**But there is no need to use new Object().**

**For readability, simplicity and execution speed, use the object literal method.**

**This Keyword :**

IN Js , the This keyword referes to an **object** .

**Which** object depends on how this is being invoked (used or called) .

The **this** keyword referes to diffrents objects depending on how it is used :

\*\* In an object method, this refers to the object .

let example = {

  userName: "Bilal",

  userLastName: "Elemrani",

  sayHello: function () {

    return "Hello " + this.userName + " " + this.userLastName;

  },

};

console.log(example.sayHello()); // Hello Bilal Elemrani

\*\* Alone, this refers to the **global object** . (window)

console.log(this); // window{..}

console.log(this === window); // true

\*\* In a function, this refers to the global object .

function hello(x) {

  return `hello ${x} ${this}`;

}

console.log(hello("Bilal")); // hello Bilal [object Window]

\*\* in a function, in strict mode, this is undefined.

"use strict";

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

\*\* IN an event, this refers to the element that received the event

document.getElementById("NF").onclick = function () {

  console.log(this);

};

// on click in the console : <button id="NF">Click</button>

***Note :***

This is not a variable. It is a keyword , you cannot change the value of this

**Create Object With Create Method:**

The object.create() method creates a new object, using an existing object as the prototype of the newly created object .

**Example :**

let user = {

  age: 40,

  doubleAge: function () {

    return this.age \* 2;

  },

};

I’ll use this as a prototype for « copyObj » object

let copyObj = Object.create(user);

When i log in the console the copyobj.age i get 40 in spite of there is no propertie name age in « copyObj » object

console.log(copyObj.age); // 40

but i can update it :

copyObj.age = 17;

console.log(copyObj.age); // 17

and sure the method work good also :

console.log(copyObj.doubleAge()); // 32

but make in mind the prototype method , « this » it should be used instead for the real name of object . because if you use in prototype method the name of protype object , the result will be : 40\*2 = 80.

And sure you can use it simply for create object :

let Obj = Object.create({ name: "Bilal" });

console.log(Obj.name); // Bilal

**Create Object With Assign Method:**

The Object.assing() method copies all properties from one or more source objects to a target object. It returns the modified target object .

**Syntax :**

Object.assign(target, ...sources)

let obj1 = {

  name: "Bilal",

  lastName: "Onahi",

  number: 18,

};

let obj2 = {

  lastName: "Elemrani",

  age: 17,

};

let finalObj = Object.assign(obj1, obj2);

//{name: 'Bilal', lastName: 'Elemrani', number: 18, age: 17}

I can add many sources, and sure the last who has the large autorithy

let obj1 = {

  name: "Bilal",

  lastName: "Onahi",

  number: 18,

};

let obj2 = {

  lastName: "Elemrani",

  age: 17,

};

let obj3 = {

  lastName: "Hakimi",

};

let finalObj = Object.assign(obj1, obj2, obj3);

//{name: 'Bilal', lastName: 'Hakimi', number: 18, age: 17}

You can also type sources and/or the target directly :

let finalObj = Object.assign({ name: "Ayoub", rank: 3 }, obj1, obj2, obj3);

//{name: 'Bilal', rank: 3, lastName: 'Hakimi', number: 18, age: 17}

**From Assingnment :**

The **Object Keys()** mehtod is used to return the object preperty name as an array .

***Simple Example :***

let user = { name: "Bilal", number: 16, class: "1Bac SMF" };

console.log(Object.keys(user)); // ['name', 'number', 'class']

The **Object.has.Own()** method returns *True* if the specified object has the indicated preperty as its own preperty. if the property is inherited, just in prototype or does not exist, the method returns *false*

let user = { name: "Bilal", number: 16, class: "1Bac SMF" };

console.log(Object.hasOwn(user, "number")); // true

let dUser = Object.create(user);

console.log(Object.hasOwn(dUser, "name")); // false

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwn#syntax)

hasOwn(object-to-test , ‘’ property for testing ‘’)

DOM

*The* ***D****ocument* ***O****bject* ***M****odel (****DOM****) is a programming interface for web documents. It represents the page , when you load your page the browser create a model for the whole page as a project that contain all element that you can behave with them .*

**1- Select Elements**

This is the first Html file that we gonna do exprience on :

<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>

  </head>

  <body style="font-size: 35px">

    <span class="my-span">My Span</span>

    <p>Hello Paragraph 1</p>

    <p>Hello Paragraph 2</p>

    <div id="my-div">Hello Div</div>

    <form action="">

      <input type="text" name="one" value="Hello" />

    </form>

    <form action="">

      <input type="text" name="two" value="Hello" />

    </form>

    <a href="https://google.com">Google</a>

    <a href="https://Twitter.com">Twitter</a>

    <script src="main.js"></script>

  </body>

</html>

**A – Find Element By ID**

The most common way to access an *HTML* element is to use the **id** of the element .

**The getElementById() method returns an element with a specified value.**

**The getElementById() method returns null if the element does not exist**.

let myDiv = document.getElementById("my-div");

console.log(myDiv); // <div id="my-div">Hello Div</div>

**Note :**

*Any* ***Id*** *should be unique, but :*

*If two or more elements with the same id exist,* ***getElementById()*** *returns First .*

**B – Find Elements By Tag Name**

The getElmentsByTagName() methods returns a collection of elements with a given tag name

The getElmentsByTagName() methods returns a NodeList object .

**-----NodeList**

A NodeList is an array-like collection(list) of nodes

*The nodes* is the list can be accessed by index, index starts by 0.

*The length* property retunrs the number of nodes

**Note :**

The Tag name « \* «  returns alll child elements

let myTagElements = document.getElementsByTagName("p");

console.log(myTagElements); // HTMLCollection(2) [p, p]

i can use indexing to access to any of the paragraphes existed in nodeList as a node .

console.log(myTagElements[1].innerHTML); // Hello Paragraph 2

wild card ‘ \* ’ to select all elements

let myTagElements = document.getElementsByTagName("\*");

console.log(myTagElements); //

//HTMLCollection(20) [html, head, script, meta, meta, link, meta, title, body, span.my-span, p, p, div#my-div, form, input, form, input, a, a, script, viewport: meta, my-div: div#my-div, one: input, two: input]

-----------------------All elements

**C – Find Elements By Class Name**

The **getElementsByClassName()** method returns a collection of element with the givin class name .

let myClassElements = document.getElementsByClassName("my-span");

console.log(myClassElements); //HTMLCollection [span.my-span]

**D – Find Element By CSS Selectors**

The **querySelector()** method returns the first child element that matches a specifed css selector of an element

The selectors can be :

class – id – tag – attributes ... (all css selectors ).

let myQueryElements = document.querySelector(".my-span");

let myQueryElementsTwo = document.querySelector("#my-div");

console.log(myQueryElements.innerHTML); //    My Span

console.log(myQueryElementsTwo.innerHTML); // Hello Div

but imagine that we have 2 span in Html , span 1 and span 2

let myQueryElements = document.querySelector(".my-span");

console.log(myQueryElements.innerHTML); // My Span 1

as you see the first span that appear.

---- Note

The querySlector() method returns the first element the specifed selectors. To returns all the matches, use **querySelectorAll()** method insead

let myQueryElementsAll = document.querySelectorAll(".my-span ");

console.log(myQueryElementsAll); // NodeList(2) [span.my-span, span.my-span]

**E – Find Element By Collection**

**------- Title :**

The **title** property sets or returns the title of the document.

**Syntax :**

\* Return the title property :

Document.title

console.log(document.title); // Js test

\* Sets the title property :

Document.title = newTitle

document.title = "To The Top";

console.log(document.title); // To The Top

**--------Body**

The **body** property sets or returns the title of the document

console.log(document.body); // <body>...</body>

document.body

or

document.body= newContent

**--------------Images**

The **images** property returns a collection of all <img> elements in a doc.

console.log(document.images); // HTMLCollection [img]

using indexing to access to any of them .

console.log(document.images[0].src); // 'img src'

document.images[0].style.border = "10px solid black";

**-------------- Forms**

The **Forms** property returns a collection of all <form> elements in a doc.

console.log(document.forms); // HTMLCollection(2) [form, form]

**-----------Links**

The **links** property returns a collection of all links elements in a doc.

The links in the collection represent <a> and <area> elemnet with an href attribute .

console.log(document.links); // HTMLCollection(2) [a, a]

and sure you can do many things . (wait we gonna learn more )

console.log(document.links[1].href); // https://twitter.com/

**InnerHTML VS InnerContent**

Have a look to this :

<div id="my-div">Hello Div &lt;span&gt;</div>

--The **innerHTML** property sets or returns the HTML content (which is inside tags in HTML document .

console.log(document.querySelector(".para").innerHTML); // Hello My Friend

as we sad you can set it  :

myDiv.innerHTML = "I am <span> Bilal </span> .";

and sure this will create a child in myDiv « span «

-- The **textContent** property sets or returns the text content of the element.

* ***Note :***

When you set the textContent property, all child nodes are removed and replaced by only one new text node

Now we will log this :

<div id="my-div">Hello Div &lt;span&gt;</div>

By innnerHTML and innerContent :

console.log(document.querySelector("#my-Div").innerHTML); //Hello Div &lt;span&gt;

console.log(document.querySelector("#my-Div").textContent); //Hello Div <span>

I think now we understand diffrence !!!!!

And the next code will show you all the diffrence

***HTML***

 <div id="my-div">

       Hello Div <span>Hello<p> evryone </p></span>

    </div>

***Js***

console.log(document.querySelector("#my-Div").innerHTML);

//       Hello Div <span>Hello<p> evryone </p></span>

console.log(document.querySelector("#my-Div").textContent);

//  Hello Div Hello evryone

console.log(document.querySelector("#my-Div").innerText);

//   Hello Div Hello

// evryone

**Get & Set attributes :**

**a- get & set attriblutes directly**

document.images[0].className = "img1";

document.images[0].title = "picture";

document.images[0].alt = "alternate";

and you can get attributes :

console.log(document.images[0].className); // img1

console.log(document.images[0].title); // picture

console.log(document.images[0].alt); // alternate

by the same method you can set or get any attributes that you want its

document.querySelector(".para").info = "class";

console.log(document.querySelector(".para").info); // class

document.getElementById("my-div").data = 152;

console.log(document.getElementById("my-div").data); // 152

**a- get & set attriblutes With Methods :**

**---------- setAttribute**

The setAttribute() method sets a new value to an attribute.

If the attribute does not exist, it is created first .

***Syntax :***

element.setAttribute(name,value)

Return Value : NONE

document.getElementById("input").setAttribute("type", "button");

document.querySelector(".para").setAttribute("number", "102");

**---------- getAttribute**

The **getAttribute()** method returns the value of an element’s attribute

Syntax :

Element.getAttribute(name)

**Return Value :**

The attibute’s value

Null if the attribute does not exist .

document.images[0].onclick = function () {

  console.log(this.getAttribute("src"));

};

Note

For find element you must use a selector that select just one like id …

DOM [check Attributes ] :

1 – element.attributes :

The **attributes** property returns a collection of attribues in an element.

The attributes property returns a NamedNodeMap

A ***NameNodeMap*** is an array-like unordered collection of an element’s attributes.

In other words, a **NamedNodeMap** is a list of **attr objects**.

A **NamedNodeMap** has a **length** property that returns the number of attr objects (nodes) . (how many attr that elemnts own )

The nodes **can be accessed** by name or index numbers.

  <p id="myP" class="my-p" info="sentence" range="1">Hello World</p>

Js :

console.log(document.getElementById("myP").attributes);

//  {0: id, 1: class, 2: info, 3: range }\*

console.log(document.getElementById("myP").attributes.length); // 4

console.log(document.getElementById("myP").attributes[1]); //  class="my-p"

console.log(document.getElementById("myP").attributes[1].value); //  my-p

console.log(document.getElementById("myP").attributes[1].name); //  class

2 – element.hasAttribute  & element.hasAttributes  :

The **hasAttribute()** method returns true if the attribute exists, otherwise false .

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

console.log(myP.hasAttribute("class") ? "yes" : "No"); // yes

The **hasAttributes()** method returns true if the elements has at least one attr, otherwise false .

  <p>Hello World</p>

Js :

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

console.log(myP.hasAttributes() ? "yes" : "No"); // NO

but know we will assign some attr to « p » :

 <p id="myP" class="my-p" info="sentence" range="1">Hello World</p>

Js :

console.log(myP.hasAttributes() ? "yes" : "No"); // yes

3 – element.removeAttribute:

The **removeAttribute()** method removes an attribute from an element.

Return Value : NONE

console.log(document.getElementById("myP").attributes);

//  {0: id, 1: class, 2: info, 3: range }

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

myP.removeAttribute("info");

console.log(document.getElementById("myP").attributes);

//  {0: id, 1: class, 2: range }

Example DOM cheaking :

console.log(myP.attributes);

if (myP.hasAttribute("info")) {

  console.log("Found");

  if (myP.getAttribute("info") === "") {

    myP.removeAttribute("info");

    console.log("Removed");

  } else {

    myP.setAttribute("info", "New Value");

    console.log("Setting New Value");

  }

} else {

  console.log("Not Found");

}

console.log(myP.attributes);

DOM [Create Elements ] :

In All This lesson :

At end of code log your created element to follow the film from cradle to grave.

1- createElement()

The createElement() method creates an element node.

let myElement = document.createElement("div");

console.log(myElement); // <div></div>

Now we will assign to it a className :

myElement.className = "product";

console.log(myElement);  // <div class="product"></div>

**Note :**

All HTML elemnts are nodes .

Elements are nodes.attributes are nodes. Text are nodes .

Some elements contain other nodes

2- createAttribute()

The createAttribute() method creates an attribute

**Syntax**

*Element.createAttribute(name)*

------

**Return Value :** the created attribute node.

let myAttr = document.createAttribute("data-custom");

console.log(myAttr); // data-custom=""

3- setAttributeNode()

The setAttributeNode() method adds an attr or replace existing attr nodes.

**Syntax**

*Element.setAttributeNode(node)*

myElement.setAttributeNode(myAttr);

console.log(myElement); // <div class="product" data-custom></div>

**setAttributeNode()** vs **setAttribute()**

setAttributeNode() cannot set a value to an attr directly , it can only sets attr name .

in effect it’s a means of preparing an attr.

setAttribute is a quicker means of doing this, it allows to set both in the same time the name of attr and attr’s value.

4- createTextNode()

The createTextNode() method create a text node .

let myText = document.createTextNode("Product One");

console.log(myText); // "Product One"

**Return Value :** the created text node

**Syntax**

*Element.createTextNode(text)*

5- appendChild()

The appendChild() method appends a node(element) as the last child of an element .

**Syntax**

*------- Element.appendChild(node)*

**Or**

*------- Node.appendChild(node)*

**Return Value :** the appended node

let myText = document.createTextNode("Product One");

myElement.appendChild(myText);

console.log(myElement); // <div class="product" data-custom>Product One</div>

and we can append all « myElement » to body :

document.body.appendChild(myElement);

6- createComment()

The createComment() method creates a comment and return the comment node.

let myCommentA = document.createComment("This is a Comment (A)");

let myCommentB = document.createComment("This is a Comment (B)");

console.log(myCommentA); // <!-- This is a Comment (A)-->

***All what we have seen :***

let myElement = document.createElement("div");

let myAttr = document.createAttribute("data-custom");

let myText = document.createTextNode("Product One");

let myCommentA = document.createComment("This is a Comment (A)");

let myCommentB = document.createComment("This is a Comment (B)");

myElement.className = "product";

myElement.setAttributeNode(myAttr);

// Append  Comment to An element (before text node)

myElement.appendChild(myCommentA);

// Append Text To an Element

myElement.appendChild(myText);

// Append  Comment to An element (after text node)

myElement.appendChild(myCommentb);

// Append Element to Body

document.body.appendChild(myElement);

console.log(myElement)

Preactice :

//  DOM

// [Create Elements] [Practice]

for (let i = 0; i < 100; i++) {

  let myMainElement = document.createElement("div");

  let myHeading = document.createElement("h3");

  let myParagraph = document.createElement("p");

  let myDiv = document.createElement("div");

  myMainElement.className = `product${i + 1}`;

  myDiv.style.cssText =

    "width:auto;height:100px;background:gray;border-radius:10px";

  let myHeadingText = document.createTextNode(`Product (${i + 1}) Title `);

  let myParagraphText = document.createTextNode("Product Description");

  myHeading.appendChild(myHeadingText);

  myParagraph.appendChild(myParagraphText);

  myMainElement.appendChild(myDiv);

  myMainElement.appendChild(myHeading);

  myMainElement.appendChild(myParagraph);

  document.body.appendChild(myMainElement);

}

document.body.style.cssText =

  "display:flex;flex-wrap:wrap;gap:40px;justify-content:space-evenly";

DOM [Deal With Childrens ] :

1-children

The children property returns a collection of an element’s child elements.

The Children property returns an HTML Collection object.

You can reset value . using ‘innerHTML or …….’

let myElement = document.querySelector("div");

console.log(myElement.children); // HTMLCollection(2) [p, span]

console.log(myElement.children[1]); // <span>Hello Span </span>

2-childNodes

The childNodes property returns a collection (list) of an element’s child nodes.

The childNodes prperty returns a NodesList object.

The childNodes property is read-only.

You can use indexing for accessing.

Important !

childNodes returns nodes : Element nodes, text nodes, and comment nodes .

**Alternative**

*The children Property*

The children property returns elements(ignores text and comments)

HTML

<div>

      Hello Div

      <p>Hello P</p>

      <span>Hello Span</span>

      <!-- Comment -->

      Hello

    </div>

JS

console.log(myElement.childNodes);

// NodeList(7) [text, p, text, span, text, comment, text]

3-

\*\*\*\*\* firstChild && lastChild

&&

\*\*\*\*\* firstElementChild && lastElementChild

The **firstChild** property returns the first child node of a node

Property is read-only

The firstChild property is the same as childNOdes [0].

**Alternative :**

The **firstElementChild** property returns the first child elements (ignores text and comment nodes ).

The **firstElementChild** property returns the same as **children[0].**

\*\* read-only

***The same speech also for lastElementChild and lastChild***

**Events**

HTML events are « **things**» that happen to HTML elements.

When js is used in HTML page, js can « **react** » on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

**Examples :**

* An HTML web page has finished loading
* An HTML button was clicked
* When an HTML form is submitted

Often, where events happen, you may want to do something.

Js lets you excute code when events are detected.

**HTML**

<button onclick="console.log('clicked')">Button</button>

    <hr />

    <form action="">

      <input type="text" />

      <input type="submit" value="Submit Data" />

    </form>

You can use events attr

**JS**

document.querySelector("button").onclick = () => console.log("Clicked");

document.querySelector("button").oncontextmenu = () =>

  console.log("ٌRight Click");

document.querySelector("button").onmouseover = () => console.log("ٌmouse over");

document.querySelector("button").onmouseout = () => console.log("ٌmouse out");

document.querySelector("button").onmouseenter = () =>

  console.log("ٌmouse enter");

document.querySelector("button").onmouseleave = () =>

  console.log("ٌmouse leave");

onmouseleave = onmouseout && onmouseenter = onmouseover

window.onscroll = () => console.log("scrolling ....");

document.querySelector("input").onfocus = () => console.log("focused");

document.querySelector("input").d = () => console.log("blur"); // our from focus

document.forms[0].onsubmit = () => alert("Are Sure, You Want to Submit ?");

**Practice :**

**HTML :**

button{

        width: 80vw;

        display: block;

        margin: 100px auto;

        font-size: 9vw;

        font-family: cursive;

        background-color:#101820FF ;

        color: #FEE715FF ;

        border-radius: 10px;

        padding: 20px;

      }

    </style>

  </head>

    <!-- ////////////// -->

    <button id="b1">Mouse Over Me</button>

    <button id="b2">Click Me</button>

Js :

document.getElementById("b1").onmouseenter = function () {

  this.innerHTML = "Thank you";

};

document.getElementById("b1").onmouseleave = function () {

  this.innerHTML = "Mouse Over Me";

};

document.getElementById("b2").onclick = function () {

  if (this.innerHTML === "Click Me") {

    this.innerHTML = "Click Me <br> Click Again";

    return;

  }

  if (this.innerHTML === "Click Me <br> Click Again") {

    this.innerHTML = "Thank You";

    return;

  }

  if (this.innerHTML === "Thank You") {

    this.innerHTML = "Good Bye";

    return;

  }

  if (this.innerHTML === "Good Bye") {

    this.style.display = "none";

    return;

  }

};

**Validate Form Practice :**

The **preventDafault()** method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur .

**NO pramater .**

For Example, this can be useful when :

* Clicking on a « Sumbit » butoon, prevent it from submitting a form .
* Clicking on a link, prevent the link from following the URL.

Note : Not all events are cancelable. Use the **cancelable** preperty to find out if an event is cancelable.

The cancelable event preperty returns a Boolean value indication wheter or not an event is a cancelable event.

document.links[0].onmouseenter = function (event) {

  console.log(this);

  console.log(event);

  console.log(event.cancelable); // false

};

document.links[0].onclick = function (event) {

  event.preventDefault(); // Prevent the link from following the URL .

};

Practice :

**HTML & CSS**

<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>

      body {

        font-size: 35px;

      }

      form {

        display: flex;

        flex-direction: column;

        gap: 20px;

      }

      form input {

        max-width: 700px;

        padding: 10px;

        width: 60vw;

        margin: auto;

      }

      h1 {

        text-align: center;

        font-family: sans-serif;

        color: dodgerblue;

      }

      input[type="submit"] {

        background: dodgerblue;

        border-radius: 5px;

        border: none;

      }

    </style>

  </head>

  <body>

    <!-- ////////////// -->

    <h1>Form</h1>

    <form action="">

      <input type="text" name="username" placeholder="Max 10 Chars Only" />

      <input type="number" name="age" placeholder="Can't Be Empty" />

      <input type="submit" />

    </form>

    <a href="https://www.google.com/">Link</a>

    <!-- ////////////// -->

    <script src="main.js"></script>

  </body>

</html>

JS

//  DOM

// [Envents]

// - Validate Form Practice

// - Prevent Default

let userName = document.querySelector("[name='username']");

let userAge = document.querySelector("[name='age']");

document.forms[0].onsubmit = function (e) {

  console.log(e.cancelable); // true

  let nameValide = false;

  let ageValide = false;

  let ageValide18 = false;

  console.log(userName.value);

  console.log(userName.value.length);

  if (userName.value.length <= 10 && userName.value !== "") {

    nameValide = true;

  }

  if (userAge.value.length !== 0) {

    ageValide = true;

  }

  if (userAge.value >= 18) {

    ageValide18 = true;

  }

  if (nameValide === false) {

    alert("The name is invalid");

    e.preventDefault();

  }

  if (ageValide === false) {

    alert("The age is invalid");

    e.preventDefault();

  }

  if (ageValide18 === false) {

    alert("This Website is For Aduls only ");

    e.preventDefault();

  }

  if (nameValide === true && ageValide === true && ageValide18) {

    alert("Submitted successfully √ ");

  }

**Events Simulation**

Events simulation are simulation of events that you can do it yourself .

<form action="">

      <input type="text" class="one" />

      <input type="text" class="two" />

      <input type="submit" value="Sumbit Data" />

    </form>

    <a href="https://Www.google.com">Google</a>

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

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

onload = () => two.focus();

one.onblur = () => document.links[0].click();

Practice :

      <input type="text" class="one" maxlength="3" placeholder="Pin (XXX)" />

      <input type="text" class="two" maxlength="3" placeholder="Second Pin (YYY)" />

      <input type="submit" value="Sumbit Data" />

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

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

let submitButoon = document.querySelector("[type = 'submit']");

onload = () => one.focus();

setInterval(() => {

  let oneValidate = false;

  let twoValidate = false;

  if (one.value.length === 3) {

    oneValidate = true;

    // one.blur();

    two.focus();

  }

  if (two.value.length === 3) {

    twoValidate = true;

  }

  if (oneValidate && twoValidate) {

    submitButoon.click();

  }

}, 1000);

HTML & CSS

<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>

      body {

        font-size: 35px;

        padding: 50px;

      }

      h1 {

        font-family: sans-serif;

      }

      input:focus {

        border: dodgerblue 2px solid;

      }

      input {

        outline: none;

        width: 70px;

        height: 80px;

        border: 2px solid black;

        border-radius: 5px;

        font-size: 50px;

        text-align: center;

      }

      form {

        display: flex;

        justify-content: space-evenly;

        flex-wrap: wrap;

        gap: 20px;

      }

      input:last-child {

        display: none;

      }

    </style>

  </head>

  <body>

    <!-- ////////////// -->

    <h1>Code Verification</h1>

    <form action="">

      <input maxlength="1" type="text" name="num1" />

      <input maxlength="1" type="text" name="num2" />

      <input maxlength="1" type="text" name="num3" />

      <input maxlength="1" type="text" name="num4" />

      <input maxlength="1" type="text" name="num5" />

      <input maxlength="1" type="text" name="num6" />

      <input type="submit" />

    </form>

    <!-- ////////////// -->

    <script src="main.js"></script>

  </body>

</html>

JS

let field1 = document.querySelector(`[name="num1"]`);

let field2 = document.querySelector(`[name="num2"]`);

let field3 = document.querySelector(`[name="num3"]`);

let field4 = document.querySelector(`[name="num4"]`);

let field5 = document.querySelector(`[name="num5"]`);

let field6 = document.querySelector(`[name="num6"]`);

let submit = document.querySelector(`[type="submit"]`);

let inputs = document.querySelectorAll("input");

window.onload = function () {

  field1.focus();

};

field1.onkeyup = () => field2.focus();

field2.onkeyup = () => field3.focus();

field3.onkeyup = () => field4.focus();

field4.onkeyup = () => field5.focus();

field5.onkeyup = () => field6.focus();

setInterval(function () {

  if (

    field1.value.length > 0 &&

    field2.value.length > 0 &&

    field3.value.length > 0 &&

    field4.value.length > 0 &&

    field5.value.length > 0 &&

    field6.value.length > 0

  ) {

    alert("Submitted successfully √  ");

    submit.click();

  }

}, 1000);

**ClassList**

The **classList** prperty returns the CSS class names of an element .

let myDiv = document.querySelector("#my-div");

console.log(myDiv.classList);

// DOMTokenList(4) ['Bilal', 'One', 'fighting', 'Np', value: 'Bilal One fighting Np']

**Return Value :** Object .

**NOTE :**

The classList preprety is read-only, but you can use the methods listed below, to add, toggle or remove CSS classes from the list

classList Property and Methods :

***add()***  🡺 add a class to an element ( ‘new’ , ‘new’ ….. )

***contains ()*** 🡺 returns true if the list contains a class

***item()*** 🡺 returns the class at a specifed index

***length*** 🡺 returns the number of class in the list

***remove()*** 🡺 removes one or more class ( ‘to-remove’ , ……)

***replace ()*** 🡺replace a class with onother one (‘old’,’new’

***toggle ()*** 🡺 add class if it’s not existed and remove it if it’s existed

<div id="my-div" class="Bilal One fighting Np">Div With Many Classes</div>

myDiv.classList.remove("Np", "Pn");

console.log(myDiv.classList.length); // 3

myDiv.classList.replace("Bilal", "BILAL");

myDiv.classList.add("show", "Mr", "heeey");

console.log(myDiv.classList.item("2")); // fighting

console.log(myDiv.classList.contains("Mr")); // true

console.log(myDiv.classList.contains("Np")); // false

myDiv.classList.toggle("heeey");

console.log(myDiv.classList);

Practice :

HTML & CSS

<style>

      body {

        padding: 50px;

      }

      .dissappear {

        transform: translateX(-600px);

      }

      #menu {

        transition: 1s;

        text-align: center;

        border: 1px solid darkblue;

        border-radius: 5px;

        width: 250px;

        margin: 30px 0 0 0px;

        box-shadow: 8px 8px 8px 0px #8888cb61;

      }

      p {

        cursor: pointer;

        margin: 0;

        padding: 10px;

        font-family: cursive;

      }

      hr {

        padding: 0;

        margin: 0;

      }

      p:hover {

        background: darkblue;

        color: white;

      }

      button {

        background: transparent;

        border: darkblue 2px solid;

        cursor: pointer;

        padding: 10px;

        font-size: 20px;

        border-radius: 10px;

      }

      button:hover {

        background-color: rgb(247, 247, 247);

      }

    </style>

  </head>

  <body>

    <!-- ////////////// -->

    <button id="button">Drop Down Menu</button>

    <div id="menu" class="dissappear">

      <p>Select 1</p>

      <hr />

      <p>Select 2</p>

      <hr />

      <p>Select 3</p>

    </div>

JS

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

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

button.onclick = () => {

  menu.classList.toggle("dissappear");

};

**CSS Styling And Stylesheets**

myDiv.style.backgroundColor = "red";

myDiv.style.color = "White";

myDiv.style.fontSize = "30px";

myDiv.style.textAlign = "Center";

or you can use CSSText :

div.style.cssText =

  "font-size:50px;color:brown;font-weight:bold;textalign:center";

The removeProperty() method removes the specified CSS property from a CSS declaration Block

myDiv.style.removeProperty("font-weight");

myDiv.style.removeProperty("color");

The setPropety() method sets a new or modifies an existing CSS property in a CSS declaration block .

myDiv.style.setProperty("color", "dodgerblue", "important");

**Syntax :**

**SetProporty (propertyName, value,priority )**

Priority is optional , A string represent if the propety’s priority should be set to important or not .

**Notes :**

You can access to all your stylesheets by :

**Document.stylesheets** and then you can use indexing to access to any of them .

And each stylesheets contain cssRules wich they are :

Element { property-style : value ; ……} **🡸 a css Rule**

You can use cssText to return all styling inside {}

console.log(document.styleSheets[0].cssRules[0].cssText);

// div { color: green; font-family: cursive; }

console.log(document.styleSheets[0]);

**Dom [ Before, After , Append , Prepend , remove]**

The before() method inserts specified content before the selected element

The after() method inserts specified content after the selected element

let myDiv = document.getElementById("my-div");

let createdP1 = document.createElement("p");

let createdP2 = document.createElement("p");

let text1 = document.createTextNode("Hello for js 'Before'");

let text2 = document.createTextNode("Hello for js 'after'");

createdP1.append(text1);

createdP2.append(text2);

myDiv.before(createdP1);

myDiv.after(createdP2);

The prepend() method inserts specified content at the beggining of the selected element

The append() method inserts specified content at the end of the selected element

myDiv.prepend("Start ");

myDiv.append(" the end");

The remove() method removes an elments (or node) from the document .

No retrun value , no prameters

myDiv.remove();

**Dom [ Trasversing ]**

\* The **nextSibling** property returns the next sibling node : an element , a text node ,a comment node.

***(next node on the same tree level ) /\_ ! \ .***

Whitespaces between elements are also text nodes .

**- previousSibling**

\* The **nextElementSibling** property returns the next sibling elment (ignores all others ).

* **previousElementSibling**

The **parentElement** returns the parent element of the specified element .

HTML :

 <div id="my-div">

      <span class="one">One</span>

      <!-- Comments -->

      <span class="two">Two</span>

      <!-- Comments -->

      <span class="three">Three</span>

    </div>

JS :

let span = document.querySelector(".two");

console.log(span.nextSibling);

console.log(span.nextElementSibling);

console.log(span.previousElementSibling);

console.log(span.previousSibling);

console.log(span.parentElement);

// practice 'remove pop up the same idea'

span.onclick = function () {

  span.parentElement.remove();

};

**Dom [ Clone ]**

The **cloneNOde()** method creates a copy of a node, and retruns the clone .

Set the deep parameter to true if you also want to clone descendats (children ).

- False (default) : clone only the node and its attributes (no text inside) .

- True : clone the node , its attributs, and desecendants.

HTML :

 <p class="my-p" id="my-p">This Is <span>P</span></p>

    <div>Div</div>

JS :

let myP = document.querySelector("p").cloneNode(true);

let mydiv = document.querySelector("div");

myP.id = `${myP.id}-Clone`;

mydiv.appendChild(myP);

**Dom [ addEventListener ]**

Now i have a problem, I Cant attach many events to an element.

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

myP.onclick = one;

myP.onclick = two;

function one() {

  console.log("Message From click one");

}

function two() {

  console.log("Message From click two");

}

As we know function two() will overide the one but me , i want the both.Here the solution is **addEvetnListener**

The addEventListener method attaches an event to an element, or to all doucment .

myP.addEventListener("click", function () {

  console.log("Message from addEvetListener");

});

myP.addEventListener("click", function () {

  console.log("Message from addEvetListener 2");

});

As you see i had many event and using this method all of them have excuted .

Now i have to do a small practice is to on p click i have to clone it an print at the end of the page .

myP.onclick = function () {

  let newP = myP.cloneNode(true);

  newP.className = "cloned";

  document.body.appendChild(newP);

  //   newP.onclick = function () {

  //     console.log("cloned");

  //   };

};

Heey this how to select event target .

document.onclick = function (e) {

  console.log(e.target);

};

A small practice again is to print on console » i am cloned » if a click something cloned .

document.addEventListener("click", function (e) {

  if (e.target.className === "cloned") {

    console.log("i am cloned");

  }

});

So Here another big feature of addEventListener is that you can catch something that is not created yet , in the document .