Important Notes

Color code
Red => Important
Purple =>Titles
Green => Object
Yellow => Method
Green =>Question
Royal blue =>Challenges

Introduction

What is JS ?

- Scripting Or Programming Language
- Working On Client Side And Server Side
- Implement Complex Features On A Web Page By Making It Interactive

What JavaScript Can Do ?

- Dynamically Update Content
- Manipulating HTML And CSS
- Animate Images And Content And Create Image Gallery
- Manipulate And Validate Data
- Control Multimedia [Audio, Video, Etc…]
- Web Browser Games
- Mobile Apps

Work with chrome developer :
As you know we are working with console to show us (info, warnings, and errors) and we can also write on it a full js code and the result will appears immediately. So use it to check your work

Where to put the JS linking code in HTML file ?

- The optimal place is before the end of the body tag (Recommended), ex :
  <script src="main.js"></script>
  </body>
- We also can place it in head tag and use some events (such as window.onload()) to make sure that all actions will take place without any errors

What is EcmaScript6?

- Ecma is the Standards organization which put the standards for the developer how to develop the language , 6 is the version on that standard
  Follow the link (https://ecma-international.org/home/)

Let's start with syntax :

1. Comments

As you know just use the short-hand (ctrl+/), The single line comment made using '//' and the multiple line comment made by /*here we write the comment*/
Note That : We use comments to describe the mystery thing not obvious things

2. Output on Screen

You have 3 ways to give an output on screen :

- Window object => Then you have to use the method alter to give that message (window.alter(""))
  Note that => this method is not Recommended hence it stops the page

- Document object => Then you have to use the method write to add this message into the document "Document == webpage" (document.write())
  Note that => If you want to add an element to you webpage we DON'T use document.write() we use document.createelement()

- Console object => Then you have to use the method log to display that message in console (console.log())
  Important note >console object is not a js object is an object related to Web and js uses the API (Application program interface) only.
  As any object, console has much many methods:
  - Log => Display a message into console
  - Error => Display an error in the console
  - Table => You pass an array in the console and this method convert that array into a table (index-value) table
    Console.table (["rana","ahmed","esraa"]) => This will be a table in console
  - ***How to style a text in console? :***
    - We use Directive %c in console text to add this styling, ex (console.log ("Rana %c Ahmed ","color: Red; font-size: 14px")), hence (Ahmed will be styled)

3. Data types

JavaScript has several datatypes, we have a built-in method in JS (typeof()), so lets talk about those types:

- String --> (TRY THAT CODE TO CHECK DATATYPE) -> console.log(typeof "Osama Mohamed");
- Number --> console.log(typeof 5000);
- **Array but the data type of it is (Object)** --> console.log(typeof [10, 15, 17]);
- Object => console.log(typeof { name: "Osama", age: 17, country: "Eg" }); and also console.log(typeof null);
- Boolean => console.log(typeof true);
- Undefined => console.log(typeof undefined);

4. Variables

→ Variables are data containers , we are using variables to add consistency to our program

→ **How to use a variable ?** => Keyword + identifier +assignment operator + value (Example : var userName = "Rana")

→ **Can I use variable without keyword ?** => You can, but it's not recommended (due to override)

→ **What is the rule of writing an identifier ?** =>
- Don't start with number, don't use ever a special character except ($) , you can start with (Letter, _ , $) and use (numbers, letters, _ , $) normally in the middle of the name, We follow the rule of camalCase

- Don't ever use a reserved name as an identifier name

→ We can add multiple variables on the same line in this method

```
// Here both variable share the same keyword
var userName = "Rana", age = 15;
console.log(userName,age)
```

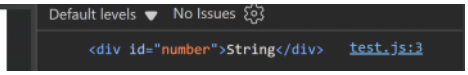
→ ID in Html file (Example :), This id is consider as a global variable that you can access it at any place and change on the element itself , **Example with code :**

- ```

      Hello

  <script src="test.js"></script>
  ```
- ```
  console.log(number)
  number.innerHTML= "String"
  ```

- String



→ **What is the difference between Loosely Typed vs Strongly Typed languages ? =>**

- Loosely Typed Languages:
  1. Variables can change type at runtime without error.
  2. Type conversions (like string to number) happen automatically (implicit type coercion).
  3. Less strict: allows mixing types in operations (e.g., adding a string and a number).
  4. Easier and faster to write initially, but bugs related to unexpected types can occur.
  5. Example languages: **JavaScript, Python, PHP.**

- Strongly Typed Languages:
  1. Variables have a specific type that must be respected.
  2. Type conversions must be explicit (you must manually convert types).
  3. More strict: type mismatches usually cause compilation or runtime errors.
  4. Safer and clearer: reduces bugs related to type confusion.
  5. Example languages: **Java, C#, Swift.**

→ **What is the difference between var , let, and const ?**

Keyword	Redeclaration	Reassignment	Access Before Declare	Variable Scope Drama such as shadowing , etc.	Scope
Var	Yes	Yes	Undefined	Yes, because each variable declared using var , is inserted into window object ,	function scope NOT Block scope
Let	NO (Error)	Yes	Error	No	Block scope
Const	NO (Error)	No	Error	No	Block scope

→ **How to write a string and what is Character Escape Sequences?** =>
We are writing a string between a "double quotes" or a 'single quotes'
What if you to write a special character that will cause a problem in string?
Just scape it using **** and to add a new line write **\n**

→ **What is the opinion of Es6 in thin way of writing ?**
We follow the concept of Template Literals (Template Strings), here we use backtick (``) and write whatever we want
Example: var userName = "Ahmed"; console.log(`${userName}`) , here we can concatenate using variables itself and no need for (+) which is the concatenation operator in JS

- We can also write a whole HTML elements within the backtick ` `

```
var content = `<div> <h1> Rana Ahmed </h1> <p> It's totally okay</p> </div>`
```

→ Here is the link of the challenge (Variable and concatenation challenge) -->
(https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Variable_Concat_Challenge.js)

5. Arithmetic Operators

→ + Addition
Normal add between 2 numbers , **what if there is a string(letters or numbers) ?** Hence concat happened example :

```
var a = "10";
var b = 20;
console.log(a + b); // 1020
```

→ - Subtraction
Normal subtraction happened , **what if there is a string (letters)?** Hence NAN appears , example :

```
var a = "RANA";
var b = 20;
console.log(a - b); //NAN
```

What I if there is a number string -> "10", hence (**type coercion happened**)

```
var a = "10";
var b = 20;
```

```
console.log(a - b); //-10
```

→ * Multiplication -> normal MUL
→ / Division -> Normal Division
→ ** Exponentiation (ES7) --> Power ex: (2**3 = 8)
→ % Modulus (Division Remainder) --> ex: (25%3 = 1)
→ ++ Increment [Post / Pre]
pre == before (Do the operation first then print the variable (example : ++x ==> increment x then print it))
post == After (print the variable first then increment it (example : x++ ==> print X then increment it))
→ -- Decrement [Post / Pre] => Same as increment

6. Unary (plus and Negation)

→ + Unary Plus [Return Number If It's Not Number]
→ - Unary Negation [Return Number If It's Not Number + Negates It
→ So to cut it short (It Convert any data type into a number (Positive or Negative) if POSSIBLE)
→ Number () Is doing the same function

7. Assignment Operators

→ We have number of assignment operators == the number of operators
Example : a += 50 --> a = a + 50
a -=50 --> a = a -50 and so on
→ Here is a link for operator challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Operators%20Challenges.js)

8. Numbers

→ Numbers in JS has complicated method to deal with, you just have to know those information :
- Syntactic Sugar "_" => This is a way to improve the appearance of syntax (Example : console.log(1_000_000) // This equal to 1000000)
- We can also use e to write whatever number of zeros we need (console.log(1e6) // This equal to 1000000)
- Power also can lead to the same result console.log(10**6) // This equal to 1000000

→ **How can I do operations on numbers ?**
- We have several ways :

1. toString() => This Method used to convert a number into a string (ex: (100).toString() // 100(String "black color on console "))

2. toFixed() => This method used to control the number of numbers after the decimal point , (ex: (120.859).toFixed(2) // 120.86)
Important Note : This method returns a STRING

3. ParseInt() => This Function take whatever you want to parse as a parameter and convert it into a INTGER number
(ex: parseInt("100 rana") // This will give 100 number)
Important note => **What is the difference between (parseInt , + unary plus, Number object)?**

Method	Example	Output	Explanation
ParseInt()	Console.log(parseInt("100 Apple"))	100	ParseInt method search for a number and then return it , and note that it can find the number if it located at the first ONLY
+ Unary plus	Console.log(+"100 Apple")	NAN	+ Unary plus can't search for a number within a string
Number()	Console.log(Number("100 Apple"))	NAN	Number object can't search for a number within a string

4. ParseFloat() => This Function take whatever you want to parse as a parameter and convert it into a FLOAT number
(ex: parseFloat("100.658 rana") // This will give 100.658 number)

5. **Number** Object => This object has multiple methods inside him, So let's talk about those methods:

- Number.MAX_SAFE_INTEGER() => This method is used to help you knowing what is the max-safe number in js , the number you can use without causing any incorrect calculations

- Number.MAX_VALUE() => This method used to tell you what is the biggest number in JS generally.

- Number.isInteger() => This method is used to tell you what you pass between () is an integer or Not (The output will be "True/ False")

- Number.isNaN ()=> This method is used to tell you what you pass between () is a NAN or Not (The output will be "True/ False")
  Note that => The condition related to scenarios where NAN appears not the appears of string

6. Math Object => This Object has several methods inside it, So let's talk about them:
   - Math.round() => This method is used to give us the closest integer number base on the numbers after the decimal point
     Ex (Console.log (Math.round(100.5)) // This will give 101) , (Console.log (Math.round(100.4)) // This will give 100)

   - Math.ceil() => This method is used to give us the number after what we write
     Ex (Console.log(Math.ceil(100.1)) // This will give us 101)

   - Math.floor() => This method is used to give us the number what we write
     Ex (Console.log(Math.floor(100.9)) // This will give us 100)

   - Math.min() => This method is used to detect the minimum number between a list of numbers
     Ex (Console.log(Math.min(1,2,3,4,8,9,80)) // This will give us 1)

   - Math.max() => This method is used to detect the maxmuim number between a list of numbers
     Ex (Console.log(Math.max(1,2,3,4,8,9,80)) // This will give us 80)

   - Math.pow() => This method is used to raise a number to some power
     Ex (Console.log(Math.pow(2,4)) // This will give us 16)

   - Math.random() => This method every time you refresh the webpage will give you a random number and later you can control that the number became random but within some list
     Ex (Console.log(Math.random()) // This will give us random number)

   - Math.trunc() => This method is used to give us the integer part only from a number
     Ex (Console.log(Math.trunc(859.65)) // This will give us 859)

7. This is the link of Numbers Challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Number%20Challenge.js)

9. String

→ To deal with string we have several methods, Now we are going to talk about each one :

- **Chain Methods :**
  - This concept is very important, hence you can do multiple operations using string methods in one line
  Example:

```
console.log(theName.trim().charAt(2).toUpperCase());
```

- How to access a string ?
  We have 2 methods :
  1. Access by using index (Normal way),
  Example :

```
let theName = "Ahmed";
console.log(theName[1]); // here will give us h
console.log(theName[5]);  //Here will give us undefined
```

  Note that : When we tried to access character that Is not included in string that will give us (Undefined)

  2. Access using charAt() => Using this method to fetch a character based on index

```
console.log(theName.charAt(1)); // Here will give us h
```

```
console.log(theName.charAt(5)); // here will give us Nothing
```

Note that : charAt() Does NOT give us undefined such as normal access using index

- Length => This method is used to return the string's length
```
console.log(theName.length); // This will return 5
```
- Trim () => This method is used to delete the spaces from the start and the end of the string
```
console.log(theName.trim()); // In case there are spaces on theName (from start and the end) will be removed
```
- toUpperCase() => This method is used to convert all string's letters into capital letters
```
console.log(theName.toUpperCase()); // AHMED
```
- toLowerCase() => This method is used to convert all string's letters into small letters
```
console.log(theName.toLowerCase()); //ahmed
```
- indexOf(value , start[opt] default 0) => This method is the opposite of charAt() , Here you pass the character itself and wait from the program the index of that char ,
  Note that you can pass one char or substring , indexOf takes [The value (اجباري), Staring position to start searching with (اختياري) and the default value is 0 (Start from the first)] , it can accept negative value (counting from end)
```
let a = "Elzero Web School";
console.log(a.indexOf("Web")); //7
```

  Important Note => Incase what you search for is not exist the program will return (-1)
```
console.log(a.indexOf("Web", 8)); //-1
```

- lastIndexOf(value, start [opt] length) => This method is same as indexOf() but here we start searching from the end and return the number of index as usual
  Example
```
console.log(a.lastIndexOf("o")); // 15  لاحظي اننا بنعد من الاخر اه ولكن لما بنرجع الرقم بنشوف الحرف دة في الطبيعي مكانه رقم كام و بنرجعه
```
- slice(start, end [Not included]) => This method is used to extract a substring from a string, This method takes (start (which is index), end (option & **not included**))
  Important Notes :
  - The start must be bigger than the end otherwise, it will give you nothing
  - If you write a negative index that means the counting will start form the end and -1 , -2, -3 and so on
```
console.log(a.slice(-5, -3)); //ch
```
- Repeat(times) => This method is used in case you want to repeat something number of times and of course take the number of repeating
```
console.log(a.repeat(2)); //Elzero Web School Elzero Web School1
```
- Split (separator, limit) => This method is used to divide the string based on your needs (بتقصقصه و بتحطه في array) , this method takes the sperator , and the limit of div idsion you want an both are optional
  Note that the return of these method is (array => which is object datatype)
```
console.log(a.split(" ",3)); //[ 'Elzero', 'Web', 'School' ]
```
- Substring(start, end [Not included]) => This method is also used to extract substring such as Slice () , what is the difference?
  - Both take the start and the end (which is Not included) But :
  1. Substring () can swap between start and end in case the start > end , Slice in that case will give you nothing
  2. Substring () Don't accept Negative index and consider this -ve (zero)
```
console.log(a.substring(-10, 6)); // 0 - 6
```

- Substr(Start [Mand], Characters To Extract) => Used to extract substring , Characteristics :
  1. takes start index , and the number of characters wanted
  2. in case the start greater than or equal the length => the output will be ""
  3. Negative start , counting form the end

- Includes (value , start [opt] default 0) => we used this method when checking if sting is exist in variable or not
  The output became (True/ False)

- StartsWith(value , start [opt] default 0) => we used this method when checking if sting starts with some character or not
  The output became (True/ False)

- endsWith(Value [Mand], **Length** [Opt] Default Full Length) [ES6] => we used this method when checking if sting ends with some character or not

The output became (True/ False)

→ This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/String%20Challenge.js)

10. Comparison & Logical Operators & If Statement & Switch statement

1. Comparison Operators :

- == -> Equal (Compares the Values ONLY)
- != -> Not Equal
- === -> Identical (**Compares both Value and dataType**)
- !== Not Identical
- > Larger Than
- >= Larger Than Or Equal
- < Smaller Than
- <= Smaller Than Or Equal

Important Question => How to have a true output from that code without change the value or change the operator

```
console.log("Osama" === "Ahmed");
```

Solution:

```
console.log(typeof "Osama" === typeof "Ahmed"); // Output => True
```

2. Logical operators

→ **! Not** --> Invert the state of variable (True -> False) , (False -> True)

→ **&& And** --> The final result will be true **ONLY when All** conditions became True

→ **|| Or** --> The final result will be true **ONLY when at least one** condition became True

3. If Statement

□ Each program is executed in normal sequence (From the top to bottom), In case you want to control that flow then use one of control flow statements , Like IF

□ If Statement :

Syntax

```
if (Condition) {
    // Block Of Code
  }
```

- Based on that conditions the block of code will be executed or not,

What if that condition is false and you what to check another condition ?

Hence, Use Else if , else

```
if (Condition) {
    // Block Of Code
  }
else if(another_condition){
    // This block will be executed in case
    //  if condition is false
}
else(){
    //  This block will be executed in case
    //  All past conditions are FALSE
}
```

- Always remember , You can use if inside another if (**Nested If**) ,and we use it in case there is a complicated logic otherwise we don't

4. Conditional Ternary Operator (Easy If)

- Syntax : Condition ? If True : If False
  Example : theName === "Rana"? Console.log(`Hi ${theName}) : Console.log("Hi")
- We can also write an if, else if, else combination:

```
theAge < 20 // if (theAge<20)
    ? console.log(20)  // In case if true
    : theAge > 20 && theAge < 60 // Else if
    ? console.log("20 To 60") // In case else if true
    : theAge > 60 // else if
    ? console.log("Larger Than 60") // incase 2'nd else if true
    : console.log("Unknown"); // incase 2'nd else if false == else
```

5. Logical Or& Nullish Coalescing Operator

- Imagine that , there is a little possibility that your data being missing in the webpage ex: price = 0 !!

Which does NOT make any sense , So using the following techniques we put a pack up in case that scenario happed

- Logical Or || : Null + Undefined + Any Falsy Value
  Example :

```
let price = null; // this is null variable
let name // This Undefined variable
let age = 0 // This Falsy value (0, "", false, etc.)
console.log(`${price||200} ${name||"Rana"} ${age||24}`) // Output 200 Rana 24
```

- Nullish Coalescing Operator ?? : Null + Undefined ONLY (Will take the falsey value as it is and put it in normal way)

```
console.log(`${price??200} ${name??"Rana"} ${age??24}`) // Output 200 Rana 0
```

- This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/If%20Condition%20Challenge.js)

6. Switch Statement

→ This statement also controls the flow of the program,

What is the difference between If and switch?

- If statement -> Mainly used with Boolean variables (اللي الاجابة بتكون فيها يا اه يا لا و على اساسه هاخد القرار) and with much more complicated decisions
- Switch Statement -> Used in Scenarios where many cases are exist "

→ **Syntax** : This expression == to what we what to take a decision based on (غالبا بيكون المتغير اللي انتي معرفاه),
case "Somthing" that thing is the option the variable might be (هو عبارة عن الاحتمالات اللي عندنا و اللي على اساسها بشوف المتغير مساوي للاختيار ف هنفذ الكود دة)

```
switch(expression) {
    Case 1:
      // Code Block
      break;
    Case 2:
      // Code Block
      break;
    Default:
      // Code Block
  }
```

→ **How switch statement comparison between the expression and case ?**
- This is happing using (Identical ===)

→ **What is the purpose of break ?**
- To stop the execution of the statement

→ **Where should I put default ?**
- Any place, It doesn't matter just don't forget break;

→ Note that : in case you have two cases will execute the same block just write for example:

```
case 2:
  case 3:
    console.log("Monday");
    break;
```

→ This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Switch%20And%20If%20Condition%20Challenge.js)

11. Array

□ **How to Create an array ?**
- We have 2 methods : 1. new Array () "This method is NOT Recommended" , 2. Using Square brackets []

□ **How to Access an array ?**
- The same method of accessing the string (Using index and we are 0-based index)

□ **Can I create a nested arrays ?**
- Sure, EX: Let myArray = [1,2,8,5,[7,9]] // To access 9 we write (myArray[4][1]) //9

□ You can also change in the array's element ex : myArray[1] = "Rana" // then myArray became : [1, 'Rana', 8, 5, [7, 9]]

□ **How can I make sure that is an array ?**
- Use Array.isArray(), Array is a constructor used to check if that is array or not

□ **Array Methods :**

- Length => This method is used to know the array's length, Always remember (The length is always Bigger than the last index in array by 1)
  - You can add , change in the array's items and control the length of it (myArray.length = 2;) // Hence myArray became [1,2]

- Unshift() => This method is used to add an element/ another array to the **FIRST** of the array

- Push () => This method is used to add an element/ another array to the **LAST** of the array

- Shift () => This method is used to remove ONE element/ array from the **FIRST** of the array and Return it

- Pop() => This method is used to remove ONE element/ array from the **LAST** of the array and Return it

- indexOf(value to search for , start (opt) default 0) => This method as you know used to search for the index of some element,
  Important Notes : Incase that element is not exist then the return of the method = -1
  In case the start in -ve number -> search from the last (right to left)

- lastIndexOf() => Same concept of indexOf() But here the default is to start searching form last (right to left)

- Includes() => This method used to check if some element is exist or not (The output is Boolean)

- Sort() => This method is used to sort the array based on alphabetic (1,2,3, and the number of zeros (من الاقل للاعلى), then a,b,c)

- Reverse() => Reverse the order of the array (first element became last and last became first)

- Slice (start, end "Not included") => This method is used to take a part of some array,
  - Note That : The main Array Doesn't effected by anything, It returns a new array
  - Negative index means start from last

- Splice(Start [Mand], DeleteCount [Opt] [0 No Remove], The Items To Add [Opt]) => This method is used to remove or add an element from/to the array
  Example :

```
let myArray = [1,2,8,5,[7,9]]
myArray.splice(0,2,"Rana")
console.log(myArray); // ["Rana",8,5,[7,9]]
```

  Super important Note : This method change in the ORIGNAL Array

- Concat(array, array) => This method is used to merge array with (one or more array , element , number) and all merged in one array and **if you check with (Array.isArray()) // Will give you true**

- Join(separator) => This method used to add all elements of array together and between each element (The wanted separator)
  Super important note => The datatype of that method is (String)

```
let myArray = [1,2,8,5,[7,9]]
console.log(myArray.join("|")); //1|2|8|5|7,9
console.log(typeof(myArray.join("|"))); //string
```

□ This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Array%20Challenge.js)

12. Loops

➢ **For Loop** : The concept of looping save you from writing block of code over and over :

- Syntax

```
for ([1] ; [2] ; [3]) {
  // Block Of Code
}
```

  [1] : Is the initialization (Ex: let I = 0;) // Note that that step can be written inside the loop (Block scope) or outside the loop (Global scope)
  [2] : Condition ,(Ex: I < 5) This is the condition which controls how many times that loop will be executed (loop will continue until that condition became False) "Note that , the condition can be written inside the loop in form of if statement"

[3] : Increment (Ex: I ++) , This part responsible of keep loop moving forward, and also can be written inside the loop

- **What is the concept of Nested for loop ?**
  - This concept means loop inside another loop , at each iteration of the main loop the inner loop will be FULLY Executed

Example

```
for (let i = 0; i < products.length; i++) {
  // first product will be displayed
  console.log(`# ${products[i]}`);
  for (let j = 0; j < colors.length; j++) {
    // ALL colors will be displayed under each product
    console.log(`- ${colors[j]}`);
  }}
```

- **What else controls the loop ?**
  - Break ; -> End the loop immediately
  - Continue; -> Will skip this iteration Only and continue the loop
  - Label -> This is like label in assembly language (Ex: mainLoop: for() then I can use this label in break and continue "break mainLoop;")

➢ **While loop**

- Syntax : The main difference in the place of elements not the elements itself

```
//here we write the counter "let i = 0"
while(condition){
  // increment the counter
}
```

- Note that : While follow the same concept of for (when the condition become false the loop break)

➢ **Do … while**

- Syntax : this rule follow the concept of doing at least one time first ,… later check

```
let i = 0;  //initialization
do {
  //block of code will be executed at least one time
  i++;  // Increment
} while (false);  // Condition
```

- This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Loop%20Challenge.js)

13. Functions

→ **What are functions ?**

- The concept of function establish the concept of (DRY "Don't Repeat Yourself "), which means to write a block of code just one time and whenever you want to do same action just call the function again

→ Syntax :

```
function functinoName(parameter)
{
  // Block of code that will be executed each time we call function
}
functinoName(argument) // function call
```

→ **What are parameter and argument ?**

- Parameters are the variable that we will work on inside the function
- Arguments are the Value of those variables in sequential order , We pass those values in function calls, (Those values will substitute the place of parameters inside that function)

→ **What can I write inside the function ?**

- We can write anything you want inside the function "If statement , loop, etc."

→ **What is the role of return inside the function ?**

- Return used in case you want to return a value from function to use this value Globally.
- Important Note :- Return follows the concept of Automatic Semicolon Insertion , which means (if you wants to return something must be written on the same line of return word)

Example

```
return something; // Correct
return   // Wrong
 something
```

- Return breaks the Entire function (Which means all code below the return will be Unreachable)

→ **Can I use return inside nested functions ?**

- Yes, We can the most important Note : (Call the inner function within the blocks of the outer function)

Example:

```
function sayMessage(fName, lName) {
    let message = `Hello`; // varible in outer function Scope
    // Nested Function
    function concatMsg() { // Nested Function 2
      function getFullName() { // Nested function 3
        return `${fName} ${lName}`;
      }
      return `${message} ${getFullName()}`; // Call for function 3 in the last line of function 2
    }
    return concatMsg(); // Call for function 2 in the last line of least outer function
  }
  console.log(sayMessage("Rana", "Ahmed")); // Hello Rana Ahmed
```

→ **How can I put default values for some parameter?** // **Note that the default value if you didn't put default value is (Undefined)**

```
  function functinoName(name)
    {
      console.log(`hi ${name}`) // hi Undefined
    }
    functinoName()
```

- We have 3 ways to put default value

1., 2. Use Condition (If param === "undefined") OR Logical Or (name = name || "Some default value") **// both are nor recommended**

3. New Way (ES6) => Put default value at the same place we write the parameters

Example:

```
   function functinoName(name = "Rana")
    {
      console.log(`hi ${name}`) // hi Rana
    }
    functinoName()
```

→ **What is the meaning of Rest parameters ?**

- Rest parameter is a concept used when the developer don't know how many arguments the user will pass , … Rest parameter help them to write arguments as much as they want and save all values in the … (parameter name) :

Example :

```
function calc(...numbers) {
  let result = 0;
  for (let i = 0; i < numbers.length; i++) {
    result += numbers[i];
  }
  console.log(`${numbers}`); // an Array of the passed argument [10,20,10,30,50,30]
  console.log(`Final Result Is ${result}`); //Final Result Is 150

}
  calc(10, 20, 10, 30, 50, 30);
```

→ This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Random%20Arguments%20Function%20Challenge.js)

→ **Can I create a function without Name ?**

- Yes , and this type is called **Anonymous Function,** This type used in case the function is doing one task (Is well known) which means the name of that function is not important at all

- **How to write that Anonymous Function?**

The basic concept is exist in store that function in a variable to be able to use that function by accessing that variable
Example :

```
let calculator = function (num1, num2) { // This is an Anonymous Function (Does Not have a name)
  return num1 + num2;
};
console.log(calculator(10, 20)); // Accessing the function using variable name and our normal arguments
```

- **Can I access the anonymous function before initialize it ?**

No , You can't do that with anonymous function because it stored in a variable (You can't access variable before initialize)
But you can do that with normal Function (Which is with name)
Example:

Anonymous function

```
console.log(calculator(10, 20));  // Error => Cannot access 'calculator' before initialization
let calculator = function (num1, num2) {
  return num1 + num2;
};
```

Named function

```
console.log(calc(5,10)) // Will Give you Normal 15 (Functions are hosted)
function calc(a,b){
  return a+b;
}
```

Don't forget to write the () When calling the variable

Anonymous functions can be passed as an argument for other function
Example :

```
let nameFunction = function() {
    return "Rana";
  };
  function concatName (firstName){
    return firstName
  }
  console.log( concatName(nameFunction()));
```

- **What is settimeout() ?**

This is a function take 2 parameters (Another function , time in ms), This function is used to tell the browser that "Execute that internal function after that time 'In ms'"

```
setTimeout(function() { // If you write a name for the Function OR Not it does't matter
  // == SetTimeout(function goodName(){code}, Time)
  console.log("Good");
}, 2000);
```

→ What is **Arrow function**?

- It is a simple way to write a function and used usually in fast/ one process task
- Syntax : 1- Remove function word , 2- Remove function's name , 3- Add () with parameters or without , 4- Write '=> { Inside it we write the function's code }'

Example

```
//Anonymous Fuction
let print = function () {
  return 10;
};

/*
Arrow Function
- No function word
- No function's name
*/
let print = () =>{
  return 10;
};
```

- **Can you remove {} and return word from the arrow function ?**

Yes, You can just in case your code consists from one line only, otherwise NO, You Can Not

- **What about removing () ?**
  # You can Remove the () In 2 scenarios : 1- There is no parameter hence we put instead _
  Example : `let print = _ => 10;  // This is arrow function does not have parameter and return 10`
- 2- In case you have 1 parameter only
  Example : `let print = num => num;  // This is arrow function have 1 parameter and return it`

- This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Arrow%20Function%20Challenge.js)

14. Scope

- We have mainly 2 types of scope :
  - **Global scope :** Each variable is declared OUTSIDE any {} That means this variable is a global variable (Which means this variable can be accessible where ever you want)
    Example :
    ```
    var a = 1; // This is a global variable
    ```
  - **Local Scope :** This scope is divided into 2 types of scope (Function Scope "Within the {} of any function" (Var) & Block scope "Within the {} Of if/for/while, etc." (Let and const)), The variables which is declared inside those blocks will be accessible within this scope ONLY other wise will give Reference Error
    Example:
    ```
    function showText() {
      var a = 10; // function scope
      console.log(`Function - From Local ${a}`); // will print 10
    }
    console.log(`From Global ${a}`); // ReferenceError: a is not defined
    ```
  - **Important Note:** Var is a function scope (Not block scope) so it can be accessible within a block inside its function and the function itself
    Example :
    ```
    function showText() {
      if (true){
        var a = 10; // Var is a function scope Not Block scope
      }
      console.log(`${a}`); // will print 10
    }
    showText()
    ```
  - **What about let and const ?**
    Both are block scope, so we can't access them out there block
    Example:
    ```
    if (10 === 10) {
    // Let is a block scope
      let x = 50;
    }
    console.log(`From If Block ${x}`); // Error => ReferenceError: x is not define
    ```
  - **Lexical scope : What if you have nested scope and the wanted value is not exist in the current scope ?**
    - Hence, We search about variables from inner scope to Outer scope to find the variable
    Example : The program will search for the written variables in the current scope (Found them ? Use them: Search in outer scope till reach the global scope)

    ```
    let country = "Egypt";
    function outer() {
      let city = "Cairo";
      function middle() {
        let area = "Nasr City";
        function inner() {
          console.log(`I live in ${area}, ${city}, ${country}`);
        }
        inner();
      }
      middle();
    }
    outer();
    ```

15. Higher Order function

- **What is the meaning of Higher Order function ?**
  - Is a function that accepts functions as parameters and/or returns a function.

- **Map ()** => This is a higher order function that accept callback function as a parameter and create a new array to put inside it the result of processing the function, هي فانكشن بتلف على عناصر الاري كلها و بتطبق عليها بعض الاوامر و الناتج من الاوامر دي بيضاف في اراي جديدة
  Let's explain more :
  Syntax : map(callBackFunction(Element, Index, Array) { }, thisArg)
  - Element => The current element being processed in the array (The first element in that array).
  - Index => The index of the current element being processed in the array. (Which is at first = 0)
  - Array => The Current Array (example: newArray.map=> newArray is the array we are doing the progess on)
  - thisArg => If you use the word **this** inside the function, it will refer to the value you pass (We will explain that later)
  - **Note that => Map() Works with arrays , so if you have a string apply . Split(" ") to convert it into array and use Join() To return it into string again**

  # We can Write the map in 2 ways :
  1- Write the function inside the map()

  Example :
```
let myNums = [1, 2, 3, 4, 5, 6];
let addSelf = myNums.map(function (ele) {  // Anonymous function
  return ele + ele});
console.log(addSelf);
//  || OR
let addSelf = myNums.map((ele) => ele + ele);  // Arrow function
console.log(addSelf);
```

  2- Define the function by name (in Normal way) and then pass it to Map()
  Example:
```
function addition(ele) {
  return ele + ele;
}
let add = myNums.map(addition);
console.log(add);
```

- **Filter ()** => This method creates a new array with all elements that pass the test implemented by the provided function.
  - **Super important Note : The new created array has ONLY the elements that passed the condition inside function**
  - Let's explain more : (Same as map())
  Syntax : filter(callBackFunction(Element, Index, Array) { }, thisArg)
  - Element => The current element being processed in the array (The first element in that array).
  - Index => The index of the current element being processed in the array. (Which is at first = 0)
  - Array => The Current Array (example: newArray.map=> newArray is the array we are doing the progess on)
  - thisArg => If you use the word **this** inside the function, it will refer to the value you pass (We will explain that later)

  Example :
```
// Get Friends With Name Starts With A
let friends = ["Ahmed", "Sameh", "Sayed", "Asmaa", "Amgad", "Israa"]
    let friendsWithA = friends.filter((ele)=> ele.charAt(0) === "A"?ele:"")
    console.log(friendsWithA)
```

  Super Important Note: What is the difference between map(), and filter() ?
  - Map () => Do some operation on the array and return a array === the length of old array
  - Filter () => Check some condition (If true "Add that element to new array", false "skip") ,
  Important Note :If there's no condition (or always true) in filter(), it returns a new array identical in values to the original.

- Reduce () => This method executes a reducer function on each element of the array, resulting in a single output value.

  - Let's explain more :
  reduce(callBackFunc(Accumulator, Current Val, Current Index, Source Array) { }, initialValue)

  - Accumulator => the accumulated value previously returned in the last invocation (At first if there is not initial value this accumulator became the first element in array)

- Current Val => The current element being processed in the array (At first if there is not initial value this current became the second element in array)

- Index => The index of the current element being processed in the array. Starts from index 0 if an initialValue is provided. Otherwise, it starts from index 1.

- Array => The Current Array

- Initial Value : Is an optional value you write in case you want to became your accumulator and first element in array became the current

Example :

```
let nums = [10, 20, 15, 30];
let add = nums.reduce(function (acc, current) {
    // acc = 10 , current = 20 => 30
    //acc = 30 , current = 15 => 45
    //acc = 45 , current = 30 => 75
  return acc + current;
});
console.log(add); // 75
```

- Foreach() => This method executes a provided function once for each array element.
  Super Important Note : Foreach () itself Doesnt Return Anything [Undefined] , which means foreach() doesn't return new array
  - Break Will Not Break The Loop

- Syntax forEach(callBackFunction(Element, Index, Array) { }, thisArg)
- Element => The current element being processed in the array.
- Index => The index of the current element being processed in the array.
- Array - The Current Array

Foreach() is most frquently Higher order function used when dealing in HTML elements in webpage (We will know this later)

16. Objects :

→ What is the meaning of an object ?

- Objects are a datatype which is most popular in programming language, This is a huge container that have 2 things :
  1. Properties => The information about that object, (you can write whatever you want)
  2. Methods => Which express the Actions that object can perform

Example :
Human => This is an object :{ // has properties like (Name, Age , Weight, Height, etc.) + Has methods (Actions) like "Moving, hearing, reading, etc."}

Code Example :

```
  console.log(window); // This is an object
   console.log(window.href) // This is a property inside window object (Return Information)
   console.log(window.location.assign("https://www.google.com")) // This is a nested object "window is an object and location is an object , assign is a method inside location object that change (ACTION) the location of the URL"
```

→ How to write an object (Syntax) ?

- // User is the object name

```
  let user = {
    // theName is a property
    theName : "Rana",
    // theAge is a property
    theAge : 24,
    // feeling is a method
    feeling: function(){
      return `Feel Nothing`
    }
  }
```

→ **How to access an object ?**

- We have 2 ways :

1. Dot notation -> This Works in case your property name follows the rules of (Identifier Naming), Otherwise it will Not work

Example :

```
// This is normal way to access an property using the concept of Dot notation (theName follow the rules)
  console.log(user.theName); // Rana
```

- **What if the property/ method name does Not follow the rule ?**

```
 let dotNotation = {
    "print me" : 50,
  }
    console.log(dotNotation.print me) //  this will throw a syntax error , "print me" => does not follow the
rule
```

- **How to solve that issue ?** => By using way number 2

2. **Bracket notation** -> This will works with both type of naming (Which follow the rules and which doesn't)

Example :

```
  let bracketNotation = {
    myName : "Rana",
    "print me" : 50,
  }
    console.log(bracketNotation["myName"]) // We put the property/ method name between "" even if it follow
the rules
    console.log(bracketNotation["print me"])  // Normally works
```

- **Super important Note => Bracket notation is the ONLY way of accessing that will work with dynamic properties**

Example :

```
  var favSong = "song"  // the value === to the property name in object
  let bracketNotation = {
    myName : "Rana",
    song : "The world is not perfect"
  }
    console.log(bracketNotation[favSong])
```

→ **Can I create nested object ?**

- Yes , without any problem

Example :

```
 let user = {
    theName : "Rana",
    theAge : 24,
    // Nested object
    hoppy: {
      music: "The world is not perfect",
      // Nested Nested object
      movies:{
        romance : "how to hate a guy in 10 days",
        secineceFaction: "Harry potter"
      }
    },
    feeling: function(){
      return `Feel Nothing`
    }
  }
  console.log(user.hoppy.movies.romance); // how to hate a guy in 10 days
  console.log(user["hoppy"].movies["secineceFaction"]); // Harry potter (Mix between dot & bracket)
```

→ **How to use property value inside a method within the same object?**

- You have to write the property in the normal way of accessing it (object name . Property name) // Don't forget () when calling the methods

Example :

```
  let user = {
    theName : "Rana",
    printName : function(){
      // you have to access the property as usual
      return`Hello ${user.theName}`
```

```
    }
  }
  console.log(user.printName()); // Hello Rana
```

→ How many ways I can create an object with ?

1. using the regular way : ex => let user = { // write properties and methods here }
   Note that you can (add and modify the values of properties / methods outside the object "Like arrays")
   Example :

```
let user = {
    theName : "Rana",
    theAge :15,
  }
  console.log(user.theAge); // 15
  user.theAge = 24;
  console.log(user.theAge); // 24
```

2. Using new Object () : This way also create an object, and between those **()** you can add whatever you want (Property/Method)
   Example :

```
let user = new Object() // empty object
  let data = new Object({  // Object by new Object has data
    theName : "Rana",
    theAge :15,
  })
```

3. As we learn Object() is an **object constructor**, By using this constructor we can create object also , let's talk about how :
   1. Object.Create() => This take a prototype (اوبجيكت يقلده) or take {} and this will create empty object
   Example :

```
let user = new Object({  // Object by new Object has data
    theName : "Rana",
    theAge :15,
  })
  let newUser = Object.create(user) // Will take all data in user as a prototype
  console.log(newUser.theName); // Rana
  newUser.theName = "Ahmed" // Here I change the data coming from user to be suit the new object (newUser)
  console.log(newUser.theName); // Ahmed
```

   2. Object.assign() => This takes (The target object to copy to, Copy the values of all of the enumerable own properties from one or more source objects to a target object) and Returns the target object.

يعني اية برضو ؟

هي بتاخد اوبجيكت تنسخ فيه و بتاخد اية الحاجات اللي هتحطها في الاوبجيكت دة و بعد كدة تساوي التارجيت اوبجيكت اللي فيه كل الداتا دلوقتي للاوبجيكت الجديد بتاعك و خلي بالك ممكن التارجيت يكون فاضي في الاول عادي

Example:

```
let user = new Object({  // Object by new Object has data
    theName : "Rana",
    theAge :15,
  })
  // Step by step explaination
  // User at first has (theName, theAge)
  // We will add to it  {gradYear:2024} , {State:"Single"}
  // Then all user will be added to newUser
  let newUser = Object.assign(user,{gradYear:2024},{State:"Single"})
  // { theName: 'Rana', theAge: 15, gradYear: 2024, State: 'Single' }
  console.log(newUser);
```

→ What is the meaning of this keyword ?

- This keyword express on the Global context (بتعبر عن المكان اللي بتتذكر فيه يعني هي مالهاش قيمة ثابتة و لكن هي بتعبر عن المالك للمكان اللي اتذكرت فيه)

- **What is the meaning of the owner of it ?**

let me explain using examples :

1. // Example 1

```
//  that is the first code in js file
  console.log(this); // This express the window object
```

2. // Example 2

```
document.getElementById("rana").onclick=function(){
    console.log(this) // Here this express the html element which has id ==="rana"
```

```
    }
```

3. // Example 3

```
  let user = {
    theName : "Ahmed",
    country :"Egypt",
    info : function(){
      return `Hello I am ${this.theName} and I am from ${this.country} ` // This here express it's owner
(which is the user object)
    }
  }
```

17. DOM

→ What is the meaning of dom ?

- Document object model -> When your webpage is rendered (Loaded) the browser creates a model of your page that have all elements inside it (اول لما الصفحة بتحمل البراوزر بيعمل موديل كدة للصفحة بتاعتك و من خلاله تقدر توصل و تتحكم في اي عنصر في صفحتك "حرفيا تقدر تتحكم في كل حاجة")

→ How can I catch any element in My webpage ?

- We have several methods to do such a thing :

1. Find Element By ID -> then Use document.getElementById()

example :

```
  //1.  store the element in a variable and work with that var later

  // Here I catch an element with name id
  let myElement = document.getElementById("name")

  // 2. Or can log it directly in console
  console.log(document.getElementById("name"))
```

Important note :Here your returned array of elements will contain just one element (Id is unique within the whole page)
So when you write :

```
console.log(document.getElementById("name")[0]); // The output will be undefined (You Only have one and only
element so you have to remove [])
```

2. Find Element By Tag Name -> Then Use document.getElementsByTagName()

Example :

```
/* Html page
     <p > Rana</p>
     <p> Ahmed</p>
  */
  let myName = document.getElementsByTagName("p")
  console.log(myName)  // here will give you a list of all p element in html file
  console.log(document.getElementsByTagName("p")[0]) // Here will give you the first element Only
Note that here => We can use the concept of indexing because we may have more than one element
```

3. Find Element By Class Name -> Then use document.getElementsByClassName()

Example :

```
  let myName = document.getElementsByClassName("name")
 console.log(myName[0])  // here will give you the first element that has a class === "name"
```

4. **Find Element By CSS Selectors** -> If you the first element the browser will meet document.querySelector (بيدور على اللي انت عايزة و بيرجع او عنصر يلاقيه فيه نفس المواصفات - لاحظي انه هيرجع واحد بس حتى لو في اكتر من عنصر في نفس الصفات) then use

Example :

```
  let myName = document.querySelector(".name") // Don't forget to put . in case class/ # in case id and etc.
  console.log(myName)
```

Important note :Here your returned array of elements will contain just one element (The first element he found)
So when you write : `console.log(myName[0]) //Undefinded`

5. Find Element By Collection -> Here will return All elements that follow your needs , Then use document.querySelectorAll()

Example :

```
/* Html page
    <p class="name"> Rana</p>
    <p class="name"> Ahmed</p>
  */
  let myName = document.querySelectorAll(".name") // Don't forget to put . in case class/ # in case id and
etc.
```

```
  console.log(myName) //Return a list of all elements
  console.log(myName[0]) //<p class = "name">Rana</p> , Here for sure you can use indexing
```

6. How can I catch the document (title/ links / images / etc.):

- It is super easy, Just write

```
console.log(document.title); // Page title
console.log(document.body); // The whole content
console.log(document.forms); // All forms and you can use indexing in normal way
console.log(document.forms[0]); //will catch the first form only
console.log(document.links[1].href); // Will catch the 2'nd link and then catch the value of its href
attribute
```

→ Now, How can I know what is the content inside a HTML element ?

- We have 2 ways :

1. inner Html -> This way give you all content between the (Opening tag and closing tag) of that element

Example :

```
  let myName = document.querySelector(".name") // Don't forget to put . in case class/ # in case id and etc.
  console.log(myName.innerHTML)  // Rana <span>Ahmed</span>
```

2. Text content -> This return the text inside the tag only, so it skip any tag (He doesn't care)

Example :

```
  let myName = document.querySelector(".name") // Don't forget to put . in case class/ # in case id and etc.
  console.log(myName.textContent)  // Rana Ahmed
```

Note that, we can change both directly as we did in arrays for example :

```
  let myName = document.querySelector(".name") // Don't forget to put . in case class/ # in case id and etc.
  myName.innerHTML= "<span>This is an updated span</span>" // here the page will contain this span sentence
has (This is an updated span)
```

OR

Use content text in case you want to display the tags as they are in your webpage

```
  let myName = document.querySelector(".name") // Don't forget to put . in case class/ # in case id and etc.
  myName.textContent= "<span>This is an <hr>updated span</span>" // here the page will contian this sentence
exactly (<span>This is an <hr>updated span</span>)
```

→ Can I change the value of some attribute ?

- Sure, you can -> You can access the attributes directly whenever you catch the element

Example : In case the attribute is already exist the value will be updated, Otherwise the attribute will be added into the element with the new value

```
/* Html page
    <p id="name">Rana</p>
  */
  let myName = document.querySelector("#name");
  myName.id = "firstName" // The id of that paragraph will be changed into firstName
  myName.className = "added-class" // A class will be added to that paragraph == added class
  console.log(myName);
```

→ Is there a method that can get and set attribute ?

- Yes we have :
- getAttribute() => This method take from you the attribute and return the attribute's value

Example :

```
/* Html page
      <p id="name" class="test">Rana</p>
  */
  let myName = document.querySelector("#name");
  console.log(myName.getAttribute("class")); // test
```

- setAttribute( ) => This method take from you the attribute and the attribute's value and append those to your element

Example:

```
/* Html page
      <p>Rana</p>
  */
  let myName = document.getElementsByTagName("p")[0]; // you have to select some element beacuse without
```

```
indexing this will retun a list of elements
  myName.setAttribute("class","first-name")
  console.log(myName); // <p class = "first-name">Rana</p>
```

→ **What if I want to check some attribute first and do some operation based on result ?**

- We have some method do that thing :

1. .attributes => This method returns all attributes that element have

Example :

```
/* Html page
    <p id="name" class="first-name" aria-multiline="true">Rana</p>
  */
  let myName = document.getElementById("name");
  // Output => NamedNodeMap {0: id, 1: class, 2: aria-multiline, id: id, class: class, aria-multiline: aria-
multiline, length: 3}
    console.log(myName.attributes); // all attributes that element has
```

2. .hasAttribute("") => This method is use to check some attribute is exist inside that element or not (So it takes the attribute's name)

Example :

```
/* Html page
    <p id="name" class="first-name" aria-multiline="true">Rana</p>
  */
  let myName = document.getElementById("name");
  console.log(myName.hasAttribute("id")); //will return true
```

3. .hasAttributes() => This method is used to check if that element has ANY attribute or not (returns false only in case the element has NO attributes)

Example :

```
/* Html page
    <p>Rana</p>
  */
  let myName = document.getElementsByTagName("p")[0];
  console.log(myName.hasAttributes()); //will return false
```

4. .removeAttribute("") => This method is used to remove some attribute form an element

Example:

```
/* Html page
    <p class="name">Rana</p>
  */
  let myName = document.getElementsByTagName("p")[0];
  myName.removeAttribute("class")
  console.log(myName.hasAttribute("class")); //will return false
```

→ **Now we want to add (element / text / comment) in our webpage , How ?**

- We have several method can do that :

1. document.createElement("") => This method used to create an element in the webpage inside that ("") we write the element name we want

Example :

```
  let myDiv = document.createElement("div");
  console.log(myDiv); //will empty div
```

2. document.createComment() => This method is used to create a comment inside the ("") we write our comment

Example :

```
let mycomment = document.createComment("This is a comment"); // In that way we created comment
```

3. document.createTextNode("") => If you want to add text into your page we have to store the text first into a variable and then append that variable to the element, We create that variable using that method

Example:

```
  let myTaxtNode = document.createTextNode("This is a text");
```

4. document.createAttribute("") => Using that method, we can also create attribute and store it in a variable, (We can add this method to our other methods)

Example:

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

5. setAttributeNode() => This method used to setup that there is an attribute will be appended into that element (element. setAttributeNode(variable created using createAttribute()))
Example
```
let myAttr = document.createAttribute("data-custom");
  myDiv.setAttributeNode(myAttr)
```

→ Finally, I created all those elements/ Textnodes/ comments/, Now I need to append them into by webpage with some Hierarchical so we use
appendChild() => Using this method you can add element to other element as we did in HTML (normal addition)
Example:
```
  let myDiv = document.createElement("div");
  let mycomment = document.createComment("This is a comment"); // In that way we created comment
  let myTaxtNode = document.createTextNode("This is a text");
  let myAttr = document.createAttribute("data-custom");
  myDiv.setAttributeNode( myAttr)
  myDiv.appendChild(mycomment)
  myDiv.appendChild(myTaxtNode)
  document.body.appendChild(myDiv)
```

→ **Now we have to learn how to deal with children using DOM ?**
- We have several methods to catch some element or all content:
1. You have to catch the parent in variable , then use:
.children -> This will return all tags that inside your parent
Example :
```
// HTML Page
//<div><!-- Osama -->Hello Div<p>Hello P</p><span>Hello Span</span><!-- Comment -->Hello</div>
let myElement = document.querySelector("div");
console.log(myElement.children); //  html collection [<p> & <span>]
```

.childNodes : Returns a node list of ALL content inside the parent tag including (Comments/ tags/ text)
Example :
```
// HTML Page
//<div><!-- Osama -->Hello Div<p>Hello P</p><span>Hello Span</span><!-- Comment -->Hello</div>
let myElement = document.querySelector("div");
console.log(myElement.childNodes); // Retuns a node list [comment, text,p,span,commnet, text]
```

#.firstChild: Returns the first THING(Might be comment, text, tag) exist after the opening tag.
example :
```
// HTML Page
//<div><!-- Osama -->Hello Div<p>Hello P</p><span>Hello Span</span><!-- Comment -->Hello</div>
let myElement = document.querySelector("div");
console.log(myElement.firstChild); // <!-- Osama --> which is the first Thing after the <div>
```

.lastChild => Returns the last THING(Might be comment, text, tag) exist before the closing tag.
example :
```
// HTML Page
//<div><!-- Osama -->Hello Div<p>Hello P</p><span>Hello Span</span><!-- Comment -->Hello</div>
let myElement = document.querySelector("div");
console.log(myElement.lastChild); // text => which is Hello
```

.firstElementChild => Returns the first tag exist after the opening tag.
example :
```
// HTML Page
//<div><!-- Osama -->Hello Div<p>Hello P</p><span>Hello Span</span><!-- Comment -->Hello</div>
let myElement = document.querySelector("div");
console.log(myElement.firstElementChild); // <p>
```

.lastElementChild => Returns the last tag exist before the closing tag.
example:
```
// HTML Page
//<div><!-- Osama -->Hello Div<p>Hello P</p><span>Hello Span</span><!-- Comment -->Hello</div>
let myElement = document.querySelector("div");
console.log(myElement.lastElementChild);// <span>
```

→ **What is the meaning of an event ?**
- Events means when some action happened, then do something we write it, Note that we write event in the way of

anonymous function = function (){}
Explanation => We have many events in JS let's take a few part of them :

- onclick : When some element is clicked, we do some action
- oncontextmenu : When the user press right click the menu will appear, then we take an action
- onmouseenter : When mouse cursor go on some element
- onmouseleave: when mouse leave some element
- onload : on loading some element (or whole window object), and so on
- onscroll
- onresize
- onfocus
- onblur
- onsubmit

→ **How to write an event ?**
Example:

```
// HTML Page
//<div><!-- Osama -->Hello Div<p>Hello P</p><span>Hello Span</span><!-- Comment -->Hello</div>
let myElement = document.querySelector("p");
myElement.onclick = function(){
  console.log("We clicked the p"); // hence when we clicked the p this sentence will be displayed on console
}
```

→ **OKAY, now I want to prevent the execution on some event, how ?**
- By using `.preventDefault();` let's take an example :

```
let myElement = document.querySelector("p");
myElement.onclick = function(e){ // e here represent the event we use (which is onclick)
  console.log(e);   // here will log information about that event (PointerEvent & type:"click" and so on)
}
```

- By catching that event I can say that when something happen prevent the execution of that event :

```
let myElement = document.querySelector("p");
myElement.onclick = function(e){ // e here represent the event we use (which is onclick)
 if(2>1){
  e.preventDefault(); // here however you clicked the p nothing will happen (منعت الوظيفة الاصلية)
 }
```

→ **Do you remember objects ?**, All right we have some object is called classList
- This object as we know each object has properties so let's talk about them

1. `classList.length` => this property returns the number of classes some element has

Example :

```
// HTML Page
// <p class="one two three four five">Hello P</p>
let myElement = document.querySelector("p");
console.log(myElement.classList.length); // 5
```

2. `classList.contains("class-name")` => This property checks if some class is exist in that element or not (That returns a Boolean value)

Example :

```
// HTML Page
// <p class="one two three four five">Hello P</p>
let myElement = document.querySelector("p");
console.log(myElement.classList.contains("one")); // true
```

3. `classList.item(index)` => Here you pass the index and the return will be the class which is in that index (return class name)

Example:

```
// HTML Page
// <p class="one two three four five">Hello P</p>
let myElement = document.querySelector("p");
console.log(myElement.classList.item(2)); // three (we are zero indexed)
```

4. `classList.add(classes-names)` => by using add, you can add as much as classes you want

Example:

```
// HTML Page
// <p class="one two three four five">Hello P</p>
```

```
let myElement = document.querySelector("p");
myElement.classList.add("six","seven"); // Here we added 2 more classes
console.log(myElement.classList) // DomTokenList['one', 'two', 'three', 'four', 'five', 'six', 'seven']
```

5. classList.remove(classes-names) => by using remove, you can delete as much as classes you want

Example:

```
// HTML Page
// <p class="one two three four five">Hello P</p>
let myElement = document.querySelector("p");
myElement.classList.remove("one","two"); // Here we removed 2 classes
console.log(myElement.classList) // DomTokenList['three', 'four', 'five']
```

6. classList.toggle(classes-names) => Toggle works in inverted way (if the class is exist then will delete it), if the class is Not exist, will add it (موجود شيله مش موجود ضيفه)

Example:

```
// HTML Page
// <p class="one two three four five">Hello P</p>
let myElement = document.querySelector("p");
myElement.classList.toggle("one","two"); // Here we removed 2 classes
console.log(myElement.classList) // DomTokenList['three', 'four', 'five']
myElement.classList.toggle("one","two"); // Here added 2 classes
console.log(myElement.classList) // DomTokenList['one','two',three', 'four', 'five']
```

→ **How to style an element in JS file?**

- we have 2 ways, 1. methods that add an inline style || 2. methods that change in the CSS file itself

1. Inline style:

First, catch the element, then we follow that syntax**(element.style.propertyName(written in camelCase) = "value";)**

Example :

```
// HTML Page
// <p >Hello P</p>
  let myElement = document.querySelector("p");
  myElement.style.color = "red"
  myElement.style.backgroundColor = "black" // The property is written in camelCase way
```

 # What if I want to write all properties in one line ?

- Use cssText => using this we can write the property name in the normal way (ex: background-color)

Example :

```
// HTML Page
// <p >Hello P</p>
  let myElement = document.querySelector("p");
  myElement.style.cssText = "color:red; background-color:black" // Normal CSS syntax
```

 # What if I need to set|| Remove some property ?

- its fine, just use style.setProperty (propertyName, PropertyValue, Important(Optional))

```
Example :
myElement.style.setProperty("font-size","40px","important")
```

- To Remove a property use style.removeProperty("property-name")

```
Example :
  myElement.style.removeProperty("color"); // color will be the default (Black)
```

2. External File (ex: Style.css)

- We use the same tech in inline style but we add something :

```
  // We access the first mentioned stylesheet in html , then the first rule (Collection of styling) which is
the p{} then the rest of tech
  document.styleSheets[0].rules[0].style.color ="purple"
```

So, We have to write first document.stylesheet[index].rules[index].style.propertyName

→ We have some additional rules related to dealing with elements:

- before [Element || String] => Used to add element || String before some element (Not inside the element) Before the whole element

Example:

```
// HTML Page
```

```
//<div> Hello div </div>
  let myElement = document.querySelector("div");
  let myp = document.createElement("p")
  let pText = document.createTextNode("This is a paragraph")
  myp.appendChild(pText)
  myElement.before(myp) // Output <p>This is a paragraph</p>
                        // <div> Hello div </div>
```

- `after [Element || String]` ==> Used to add element || String after some element (Not inside the element) after the whole element

Example :

```
// HTML Page
//<div> Hello div </div>
  let myElement = document.querySelector("div");
  let myp = document.createElement("p")
  let pText = document.createTextNode("This is a paragraph")
  myp.appendChild(pText)
  myElement.after(myp) // Output <div> Hello div </div>
                       //  <p>This is a paragraph</p>
```

- `append [Element || String]` => used to add element || text inside the element and AT THE END

Example

```
// HTML Page
//<div> Hello div </div>
  let myElement = document.querySelector("div");
  let myp = document.createElement("p")
  let pText = document.createTextNode("This is a paragraph")
  myp.appendChild(pText)
  myElement.append(myp) // Output <div> Hello div
                        // <p>This is a paragraph</p>
                        // </div>
```

- `prepend [Element || String]` used to add element || text inside the element and AT THE Beginning

Example

```
// HTML Page
//<div> Hello div </div>
  let myElement = document.querySelector("div");
  let myp = document.createElement("p")
  let pText = document.createTextNode("This is a paragraph")
  myp.appendChild(pText)
  myElement.prepend(myp) // Output <div>
                         // <p>This is a paragraph</p>
                        // Hello div
                        // </div>
```

- `element.remove()` => this used to remove that element completely from the webpage

Remember: When we wanted to make some element invisible we did it using CSS=> display : none; but here the element is still in the html page

Using Remove() , it will deleted

→ **Dom Traversing** => This concept make us able to know what is the next, previous, parent elements of some element

- `.nextSibling` => used to get the next sibling (whatever what it is "text, comment, element")

Example:

```
/* HTML Page
  <div> Hello div
       <p class="p1">p1</p>
       <!-- rana -->
       <p>p2</p>
    </div>
*/
  let myElement = document.querySelector(".p1");
  console.log( myElement.nextSibling); // <!-- rana -->  which is comment
```

- `.previousSibling` => used to get the previous sibling (whatever what it is "text, comment, element")

Example:

```
/* HTML Page
  <div> Hello div
```

```
        <p class="p1">p1</p>
        <!-- rana -->
        <p class= "p2">p2</p>
    </div>
*/
  let myElement = document.querySelector(".p2");
  console.log( myElement.previousSibling); // <!-- rana -->  which is comment
```

- `.nextElementSibling` => used to get the next sibling element Only

Example:

```
/* HTML Page
  <div> Hello div
        <p class="p1">p1</p>
        <!-- rana -->
        <p class= "p2">p2</p>
    </div>
*/
  let myElement = document.querySelector(".p1");
  console.log( myElement.nextElementSibling); //<p class="p2">p2</p>
```

- `.previousElementSibling` => used to get the previous sibling element Only

Example:

```
/* HTML Page
  <div> Hello div
        <p class="p1">p1</p>
        <!-- rana -->
        <p class= "p2">p2</p>
    </div>
*/
  let myElement = document.querySelector(".p2");
  console.log( myElement.previousElementSibling); //<p class="p1">p1</p>
```

- `.parentElement` => Returns the parent of the element

Example:

```
/* HTML Page
  <div> Hello div
        <p class="p1">p1</p>
        <!-- rana -->
        <p class= "p2">p2</p>
    </div>
*/
  let myElement = document.querySelector(".p1");
  console.log( myElement.parentElement); //<di></di>
```

→ What is the meaning of DOM Cloning ?

- Is to take a copy from some element and use that copy as you want,

 # How to do that ?

We have some method called .**cloneNode**(), This method take one of 2 things (true || false)

- In case of 'True' => the created copy will have all attributes and all elements that is inside the original copy

- In case of 'False' => the created copy will have all attributes of the original copy Only and will Not have any internal content

Example :

1. False || Empty

```
/* HTML Page
   <div class="my-div"> Hello div
       <p>p1</p>
    </div>
*/
  let divCopy = document.querySelector(".my-div").cloneNode()
  console.log( divCopy); //  <div class="my-div"> </div>
```

2. True

```
/* HTML Page
   <div class="my-div"> Hello div
       <p>p1</p>
    </div>
*/
  let divCopy = document.querySelector(".my-div").cloneNode(true)
  console.log( divCopy); //  <div class="my-div"> Hello div <p>p1</p>  </div>
```

→ **What is addEventListener ?**

- addEventListener is an advanced way to add events to an element

1. What is the difference between normal way and addEventListener ?

	Normal way	addEventListener

Way of writing	Write the event with 'on'
	Example:
	myelement.onclick = function(){
	// here we write the code
	}
		We write the event direct
		Example:
		myelement.addEventListener ("click",function(){
		// here we write the code
		})

Multiple events	In case you add two event on the same element, the second one will overwrite the first which means (One event Only will be executed)	In case you add two event on the same element, both events will be execute
Writing a string instead of function	Example: `window.onload = "rana"; // This will give nothing so you can't catch the error`	Example: `divCopy.addEventListener("click", "String"); // Error => parameter 2 is not of type 'Object'.`
Add event on element before creation	ERROR	Will wait until the element be created based on the logic and then will execute the event

→ This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/tree/main/Dom%20Challenge)

18. BOM [Browser Object Model]

→ **What is the meaning of browser object model ?**

- It is the window (The current tap) object as you know

- Note that: The DOM Is a part of the BOM

example:

```
let div = window.document.getElementsByClassName("my-div") // this syntax is correct

- Everything that located in the webpage is a part of the window object and you can access it
```

→ Now, Let's talk about the Windo object 's Methods :

1. Alert () => This method is used to display a message, Until you press OK the execution of the webpage will be paused

Example :

```
window.alert("Hello") // pop out message will appear and the whole execution of the webpage will stop
```

2. Confirm() => This method displays a message and wait from the user side a response (Ok === true OR Cancel === False), The method returns a Boolean value based on user's choose

Example:

```
let bool = window.confirm("Are you okay?")
if(bool === true){
  console.log("I am happy to here that "); // when press okay
}
else{
  console.log("I feel bad for you") // when press cancel
}
```

3. Prompt (message, placeholder message) => This method returns what you wrote in the input field (Collecting data from you)

Example:

```
let age = window.prompt("How old are you","You must write a number") // Here the pop out will appear and ask me about my age with reminder to write numbers only
  console.log(`Your age is ${age}`); // Will log the written age
```

4. open(URL [Opt], Window Name Or Target Attr [Opt], Win Features [Opt], History Replace [Opt]) => This method used to open new window and it takes some parameters and all of them are optional:

- URL -> the URL of the new window (in case no param , will open empty tap)
- Target -> _blank (will open in new tap) , self (will change the exist tap)
- Win features -> you can manage some features like (left, top, width, height)

Example:

```
setTimeout(function () {
  // This will open new window in new tap with those charctristics after 2s
  window.open("https://google.com", "_blank", "width=400,height=400,left=200,top=10");
}, 2000);
```

5. Close() => This method close ONLY the window that is opened using window.open()

6. stop() => each webpage takes some time to complete rendering and loading all media and data in the page, Window.stop() stops that loading

7. Print() => This works as same as ctrl+p which is responsible for printing the page (window.print())

8. focus() => As the property in input field, concentrate the focus on some window (window.focus())

9. ScrollTo && scroll (x,y|| Options) => scrollTo() and scroll() both do the same thing it is related only to the browser support (So just check it)

These method tell the browser to go to specific place in the window

Example:

```
// This example using the x, y values
window.scrollTo(400,400) // Using that way has a poor smoothness
// This example using the Options [left. top, behavior]
window.scrollTo({
left: 500,
top: 200,
behavior: "smooth"
});
```

10. scrollBy(x,y|| Options) => These method tell the browser ON EACH REFRESH move by specific amount, so the browser with each refresh will add to the current position that amount

Example:

```
// This example using the x, y values
window.scrollBy(400,400) // Using that way has a poor smoothness
// This example using the Options [left. top, behavior]
window.scrollBy({
left: 500,
top: 200,
behavior: "smooth"
→ });
```

11. scrollX === pageXOffset => This is a property that returns the amount of scrolling in x-axis (horizontally)
Note that: Both are doing the same function but use pageXOffset whenever you are working on old browsers
Example :

```
console.log (window.scrollX) // log the x-position scrolled at that time
```

12. scrollY === pageYOffset => This is a property that returns the amount of scrolling in y-axis (Vertically)
Note that: Both are doing the same function but use pageYOffset whenever you are working on old browsers
Example :

```
console.log (window.scrollY) // log the Y-position scrolled at that time
```

13. LocalStorage:

- **What is the meaning of localStorage?**

• Let's imagine, you have to safe some settings related to your website (ex: the main color, the mode "Dark|| light", etc.) Those information need to be stored in all time for that website, even If you close the tap and open it again the data will still exist === (No Expiration Time)
What we just said is the meaning of localStorage

- LocalStorage is an object (with key and value) and we can apply all the concepts of object on it, So let's start

1. setItem() => This method used to add pair of key and value in the localStorage and we can do the same thing using the dot & bracket notations
Example:
```
window.localStorage.setItem("color","purple") // here we add a key-value pair into the object
window.localStorage.width = "400px" // Use another way of adding (Dot notation)
window.localStorage["padding"] = "15px" // Use another way of adding (Bracket notation)
```

 # To see that object on the website => Go to application and then select the localStorage

2. getItem() => This method returns the value of the passed key and also we can do the same thing using the dot & bracket notations
Example :
```
console.log(  window.localStorage.getItem("color") ); // purple
console.log(  window.localStorage.color ); // purple
console.log(  window.localStorage["color"] ); // purple
```

3. removeItem() => This method remove the item that related to the passed key (For sure remove both key and value)
Example:
```
// Now the localStorage contains 3 items color, width and padding
window.localStorage.removeItem("color") // Will remove color and the remaining are (width and padding)
console.log(window.localStorage.getItem(color)); // ReferenceError: color is not defined "
```

4. clear() => This method Clears everything (key-value pair) exist in the localStorage
Example:
```
window.localStorage.clear() // Clear Everything that exist in the localStorage === Empty LocalStorage
```

5. Key() => This method takes form you an index and give you the key which at that Index in the localStorage
Example:
```
// We have key with that sequence [color, width, padding]
console.log(window.localStorage.key(0));   // Color
```

14. SessionStorage:

- What is the meaning of sessionStorage?

• It means the data will be exist ONLY within that session == Tap
When you open a new tap === New session with new data
when you open a new tap with even the same URL of the exist session === New session with new data
when you Duplicate the tap that means === The same Session with the same data

Super Important note => All methods we talked about in LocalStorage in typically exist in the sessionStorage With no differences in the functionality, The main difference exists in the meaning of localStorage and sessionStorage

→ **What are the actions we can do and it is related to Window object ?**

1. **setTimeOut(Function, time in ms, Another parameters "If Exist")** => This is a Web API function provided by the browser (Window), that accepts 2 things mainly :
- Function -> Could be anonymous function or regular one
- Time -> The exact time after it the function will be executed (Which means the waiting time before the execution)
- Another Parameters -> In case you used a regular function that will accept parameters, then that parameters will be written in the setTimeOut()

Example :
```
// First form
setTimeout(function(){
  console.log("Rana"); // this will be logged after 1 s === 1000 ms "Rana"
},1000)

// Second Form
setTimeout(lastName,1000) //this will be logged after 1 s === 1000 ms "Ahmed"
function lastName(){
  console.log("Ahmed");
}
```

```
// Third Form
setTimeout(userAge,2000,"24") //this will be logged after 2 s === 2000 ms "Your age is 24"
function userAge(age){
  console.log(`Your age is ${age}`)
}
```

2. clearTimeOut(**Identifier**) => This built-in function stops the effect of setTimeOut(), but you have to pass the counter which is holding the setTimeOut()
Example:

```
setTimeout(function(){
  console.log("Rana"); // this will be logged after 1 s === 1000 ms "Rana"
},1000)

let counter = setTimeout(userAge,2000,"24") // This Code is NOT executable
function userAge(age){
  console.log(`Your age is ${age}`)
}
clearTimeout(counter) // Here the clearTimeOut() will stop the execution of counter variable which is setTimeOut()
```

3. setInterval(Function, Time in ms , Another parameters) => This built-in function works on Repeating the passed function over and over based on the passed Time.
Note that => The syntax of setInterval() Is exactly same as setTimeOut(), But the difference in the way of working

Example :

```
setInterval(function(){
  console.log("Rana"); // this will be logged every 1s Rana
},1000)
```

4. clearInterval(**Identifier**) => This built-in function stops the effect of setInterval(), but you have to pass the counter which is holding the setInterval()
Example:

```
let counter = setInterval(userAge,2000,"24") // This Code is NOT executable
function userAge(age){
  console.log(`Your age is ${age}`)
}
clearInterval(counter) // Here the clearInterval() will stop the execution of counter variable which is setInterval()
```

→ **What is Location Object ?**
- Location object is an object inside the window object that has some method we can use, Let's take about them:
1. Location.href => This property gets / Or sets a [URL || Hash === Specific section with ID || File || Mail] for you
Example :

```
console.log(location.href); // http://127.0.0.1:5500/
location.href = "https://www.google.com" // Will Go to Google
```

2. Location.Host => This property returns the Host, In addition to the port number
Example:

```
console.log(location.host); //  Host + port number -> 127.0.0.1:5500
```

3.Location.hostname => This property returns the host Only
Example:

```
console.log(location.hostname); // Host name only  -> 127.0.0.1
```

4. Location.hash => It gets or sets the part of the URL after the # symbol. Commonly used for navigating to sections on the same page.
Example:

```
// The URL -> https://example.com/page.html#about
console.log(location.hash); // "#about"
```

5. location.protocol => It gets or sets the protocol of the webpage
Example :

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

6. location.reload() => Reloads the current page.

7. location.replace("Take here an URL") => Replace the current URL with the written URL **+ Super important note:**
It deletes the replaced entity (page) from the history
Example:

```
location.replace("https://open.spotify.com/"); // The current tap will be replace by that URL and also will be removed from History
```

8. location.assign ("Take here an URL") => Replace the current URL with the written URL + **Super important note:**
It keeps the replaced entity (page) in the history
Example:

```
location.assign("https://open.spotify.com/"); // The current tap will be replace by that URL and also will be still exist in History
```

→ **What is History Object ?**
- This is an object inside the window object that responsible on the taps history, which one is visited first and so on.
This object has some properties, let's talk about them :
1. length -> Returns the number of websites that be visited and stored for that tap

What about methods :
- Back() => We will go to the previous page that is in the history
- forward() => We will go to the next page that is in the history
- go(index) => It does the same function on back(), forword() and also reload(). It depends on the index
0 => reload() , +ve => the next page , ex go(2) == go to the second page in history , -ve() => previous page ex go(-2) == go to the second page from the end

→ This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/tree/main/Bom%20Challenge)

19. Destructuring

➢ The meaning of Destructuring is a JavaScript expression that allows us to extract data from arrays, objects, and maps and set them into new, distinct variables.

➢ **We will Start with Destructuring Arrays:**
- First you declare the variable you want to store the extracted data in
- Second in arrays you must To determine with element you want and if you want to Skip some element we put ,
- All Operation in made in []

Let's Explain more with Example :

Example 1 : Simple Array

```
let myFriends = ["Ahmed","Sayed","Ali","Shady","Amr","Mohamed","Gamal"]
// You want to Extract "Ali" ONLY, Then:
// First declare the variable in [] because we deal with array
// You want to skip all elements ecxept "Ali"
let [, , a, , , , ] = myFriends
console.log(a); // Ali (Done)
```

Example 2 : Nested array

```
let myFriends = ["Ahmed","Sayed","Ali",["Shady","Amr",["Mohamed","Gamal"]]]
// You want to Extract "Shady" and "Gamal" ONLY, Then:
// First declare the variable in [] because we deal with array
// You want to skip all elements ecxept  "Shady" and "Gamal"
let [, , , [a , ,[ ,b]]] = myFriends
console.log(a); // Shady (Done)
console.log(b); // Gamal (Done)
```

Example 3 : Swapping

```
// Swap using Destructuring
let book = "Video";
let video = "Book";
// Here the value of book variable "Video" has been assigned to the video variable and vice versa
[book,video] = [video,book]
console.log(book);
console.log(video);
```

- **Destructuring Objects :**

- First you declare the variable you want to store the extracted data in
- Second in objects you determine the element you want by it's key and you can rename those variables also,
- All Operation in made in {}

Let's Explain with code:

```
let user ={
  firstName: "Rana",
  lastName:"Ahmed",
  age:24,
}
  // Here we extract firstName property from user object and give it another name "f"
  // Here we extract lastName property from user object and give it another name "l"
  // Here we extract age property from user object and give it another name "a"
let {firstName:f,lastName:l,age:a}= user
console.log(f); // Rana
console.log(l); // Ahmed
console.log(a); // age
```

Example 2 : Nested object

```
const user = {
  theName: "Osama",
  theAge: 39,
  theTitle: "Developer",
  theCountry: "Egypt",
  theColor: "Black",
  skills: {
    html: 70,
    css: 80,
  },
};
//In case the nested object we write the property name and then : and acess whatever the inside property
let {theName:n,theAge:a,skills:{html:h}}=user
console.log(n); //Osama
console.log(a); //39
console.log(h); //70
```

- **Destructuring Function Parameters:**

- We can apply the concept of destructuring also in function parameter in case it takes array or object

Let's explain more with example :

```
const user = {
  theName: "Osama",
  theAge: 39,
  theTitle: "Developer",
  theCountry: "Egypt",
  theColor: "Black",
  skills: {
    html: 70,
    css: 80,
  },
};
showUserInfo(user) // Osama,39,70
function showUserInfo({theName:n,theAge:a,skills:{html:h}}=user){
console.log(n);
console.log(a);
console.log(h);
}
```

- **For sure we can deal also with Mixed data**

Example:

```
// Object has inside it 1. array 2. object
const user = {
  theName: "Osama",
  theAge: 39,
  skills: ["HTML", "CSS", "JavaScript"],
  addresses: {
    egypt: "Cairo",
    ksa: "Riyadh",
```

```
  },
};
let {theName:n,theAge:a,skills:[ , ,JavaScript],addresses:{ksa}}=user
console.log(`${n} ${a} ${JavaScript} ${ksa}`);
```

→ This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Destructuring%20Challenge.js)

20. Sets

→ Sets are a data type such as arrays but has some differences:
1. Object To Store Unique Values "It can store any type of data and also can mix between the data types But" **(No Redundancy)**
2. Cannot Access Elements By Index
Let's take an example before talking about its properties and methods

Example
```
  let myArray = [1,2,2,3,4,5,5,6,1]
   let mySet = new Set(myArray) // set(6){1,2,3,4,5,6} => Set Contains the Unique Values only from the
myArray
   console.log(mySet);
   console.log(typeof mySet); //Object
```

→ How can we know the length or the size of the set ?
- We have property (size that give us the size of the set)
Example
```
  let myArray = [1,2,2,3,4,5,5,6,1]
   let mySet = new Set(myArray)
   console.log(mySet); //{ 1, 2, 3, 4, 5, 6 }
   console.log(mySet.size); //6
```

→ **How can we add elements into the set ?**
- We have multiple ways to do such a thing :

1. `let mySet = new Set([1,2,3,3,3,4,5]) // Write the values directly` **In []**

2.
```
let myArray = [1,2,2,3,4,5,5,6,1]
let mySet = new Set(myArray) // Pass the variable name
```

3. Use add() method
Example :
```
   let mySet = new Set().add(1).add(2).add(2).add(3) //Use add method
```

→ Let's Talk about the other method :
2. delete() => This method deletes an element from the set and returns a Boolean value
(True-> If the element has been found and deleted from the set, False -> In case the element is Not exist in the set)
Example :
```
  let myArray = [1,2,2,3,4,5,5,6,1]
   let mySet = new Set(myArray)
   console.log(mySet); //{ 1, 2, 3, 4, 5, 6 }
   console.log(mySet.delete(1)); // true => The element is exist (1) and has been deleted successfully
deleted
   console.log(mySet); //{2,3,4,5,6}
```

3. Clear() => This method clear the entry set (Turn it into empty set)
Example :
```
  let myArray = [1,2,2,3,4,5,5,6,1]
   let mySet = new Set(myArray)
   console.log(mySet); //{ 1, 2, 3, 4, 5, 6 }
   mySet.clear() // Clear all elements in the set
   console.log(mySet); //Set(0){}
```

4.has () => This method checks if some element is exist in the set or not (Returns a Boolean value "true || false")
Example:
```
  let myArray = [1,2,2,3,4,5,5,6,1]
   let mySet = new Set(myArray)
   console.log(mySet); //{ 1, 2, 3, 4, 5, 6 }
   console.log(mySet.has(5)); //True
```

```
console.log(mySet.has(10)); //False
```

→ **What is the meaning of weakSet ?**

- The WeakSet is weak, meaning references to objects in a WeakSet are held weakly. If no other references to an object stored in the WeakSet exist, those objects can be garbage collected.
بمعنى ان الاوبجيكتس اللي جوا الويك سيت دي لو ماتمش استخدامهم او بمعنى اصح الاشارة ليهم في الكود بيكونوا اكتر عرضة انهم يتمسحوا من الميموري

- In another meaning ->Store objects and removes them once they become inaccessible

→ **What are the differences between Set and weakSet?**

	Set	WeakSet

Store data	Store any type of data values
	`let mySet = new Set([1, 1, 1, 2, 3, "A", "A"]);`
	`console.log(mySet); //{1,2,3,"A"}`
		Collection Of Objects Only
		`let myWeakSet = new WeakSet([1, 1, 1, 2, 3, "A", "A"])`
		`console.log(myWeakSet); // Error=> TypeError: Invalid value used in weak set`
		`let myWeakSet = new WeakSet([{ A: 1, B: 2 }]) // This is the correct way`

Size property	Have Size Property
	`let mySet = new Set([1, 1, 1, 2, 3, "A", "A"]);`
	`console.log(mySet.size); //4`
		Does Not Have Size Property
		`let myWeakSet = new WeakSet([{ A: 1, B: 2 }])`
		`console.log(myWeakSet.size); //undefined`

Keys, Values, Entries	Have Keys, Values, Entries
	`let mySet = new Set([1, 1, 1, 2, 3, "A", "A"]);`
	`let iterator = mySet.keys(); // The iteratror here contians the values of mySet`
	`console.log(iterator.next().value); // will log the most first element "1"` في الاول بيكون واقف قبل خط البداية و بعد كدة بياخد اول خطوة باول عنصر
	`console.log(iterator.next().value); // will log 2 , done:false (Which means the iteration is still running)`
	`console.log(iterator.next().value); // 3 , done:false (Which means the iteration is still running)`
	`console.log(iterator.next().value); // "A" , done:false (Which means the iteration is still running)`
	`console.log(iterator.next()); // Undefinded + done:True ( Which means the iteration finished )`
		Does Not Have clear, Keys, Values And Entries

forEach()	Can use forEach()
	`let mySet = new Set([1, 1, 1, 2, 3, "A", "A"]);`
	`mySet.forEach((e)=>console.log(e)) // Iterate on all elements and log them one by one`
		Cannot Use forEach

21. Map

→ It is a data type that takes an Iterable and (key-value pair), If you want to insert elements to the map() Directly then same as sets you have to write each key-value pair In [] and the whole elements in [] also
Example:

```
let myMap = new Map();

  let myMap = new Map([
    [10, "Number"],
    ["Name", "String"],
    [false, "Boolean"],
  ]);
```

Super Important note => The previous line will give you an Object has NO Properties
You may think it's the normal thing when you declare an empty object but its NOT

→ What is the difference between the map and object ?

	Object	Map

Default key	Has Default Keys
	`let myObject = {};`
	`console.log(myObject); // this will log an empty object that is correct BUT it has a prototype and we will explain that later`
		Does Not Contain Key By Default
		`let myMap = new Map();`
		`console.log( myMap); // No properties`

	What if you want your object Initially be exactly like the map ?
	Then =>
	`let myEmptyObject = Object.create(null);`
	`console.log(myEmptyObject); // This will give you the exactly result of the creating an empty map`	

Key rules	The key can be string or symbol only
	Example:
	
	`let myNewObject = {`
	`  10: "Number",`
	`  "10": "String",`
	`};`
	`console.log(myNewObject[10]);  // This will give us string, why ? the object treate the two keys as a one key and has been overwrite`
	`console.log(myNewObject); // This will give as only one property  10:"string" => which is in his concept the overwritten value`
		Key Can Be Anything [Function, Object, Any Primitive Data Types]
		
		`// What about Map()`
		`let myNewMap = new Map();`
		`// we add to the object using set (key, value)`
		`myNewMap.set(10, "Number"); // this is 10 "number"`
		`myNewMap.set("10", "String"); // this is "10" -> string (Map doesn't mix between them)`
		`myNewMap.set(true, "Boolean");  // accept boolean`
		`myNewMap.set({a: 1, b: 2}, "Object"); // another object`
		`myNewMap.set(function doSomething() {}, "Function"); // function`
		`console.log(myNewMap.get(10)); // Number`
		`console.log(myNewMap.get("10")); // String`

Order	The order of the properties is Not 100% match the insertion order	Ordered By Insertion
Size	Need To Do Manually	Get Items By Size Property
Iterations	Not Directly And Need To Use Object.keys() And So On	Can Be Directly Iterated
Performance	Low Performance When Comparing To Map	Better Performance When Add Or Remove Data

→ **Map Properties and Methods :**

1. Property => Size exactly like sets (Give you the length or size of the map)

Example :

```
let myMap = new Map([
  [10, "Number"],
  ["Name", "String"],
  [false, "Boolean"],
]);
console.log(myMap.size); // 3 (key-values pair)
```

2. Methods :

- Set() => Take the key and the value and insert them into the map

Example:

```
let myMap = new Map([
  [10, "Number"],
  ["Name", "String"],
  [false, "Boolean"],
]);
myMap.set({age:10}, "object"); // Here I inserted a new element
console.log(myMap.size); // 4
```

- delete() => take the key and delete it from the map (if it exist) exactly like delete in set

Example:

```
let myMap = new Map([
  [10, "Number"],
  ["Name", "String"],
  [false, "Boolean"],
]);
console.log(myMap.delete("Name")); // true => The element has been found and seccessfuly delete exaclty as delete in set
console.log(myMap); //Map(2) {10 => 'Number', false => 'Boolean'}
```

-Clear() => Same as in set

- has () => Same as in set

→ **What is the difference between map and Weakmap ?**

- Shortly, It is same as the differences between set and weakset,

1. WeakMap Allows garbage collector to do it's task in case one of it's object it not used But regular Map does Not

2. Map's key => Can be anything as we said
WeakMap's key => Can be object Only

22. Array Methods

1. Array.from(iterable, mapfunction(), this "As we know if you didn't select it to some element it's owned by its owner ") => This method is used to extract an array from and iterable item.
Super Important Note => Here we use the Array Constructor
Let's take examples:

```
Example 1
console.log(Array.from("1,5,rana")); // This will give us array of [1,5,"r","a","n","a"]

Example 2
// What if I told to made a unique array from that array ?
  let myArray = [1,2,3,4,2,3,5,6,4,7]
  let mySet = new Set(myArray)
  let uniArray = Array.from(mySet)
  console.log(uniArray);

  //OR
  console.log([...new Set(myArray)]);  // In Only one step using spread operator
```

2. Some Array.copywithin(Target, Start[opt], End [opt]) => This method helps you to copy a part of your array and paste it in the target index you written
Super important note => This method is NOT increase the length of the array , we copy some element and paste them with the complete commit to the actual length
Example:

```
  let myArray = [10,20,30,40,50,"A","B"]
   console.log(myArray.copyWithin(3,0,4)); //[10,20,30,10,20,30,40]
   console.log(myArray.copyWithin(1,-2,-1)); // [10,"A",30,40,50,"A","B"]
```

3. Array.some(callbackFunction(element,index,array),this Argument)
- CallBackFunction -> Is the function to run on every element on the given array
- Element -> Presents the element be executed now
- Index -> The index of the executed element right now
- Array -> On which array we are working
- This -> Value used as this when executed the callback function
This method returns (True => In case at least on element satisfy the condition , False => In case all elements DO NOT satisfy the condition)
Example:

```
  let myArray = [10,20,30,40,50,"A","B"]
   console.log(myArray.some((e)=> +e > 40)); // true because we have 50 which is bigger than 40
   console.log(myArray.some(function(e){
     return +e > 50
   })); // False because we have don't have any element bigger than 50
```

4. Array.every(callbackFunction(element,index,array),this Argument)
- CallBackFunction -> Is the function to run on every element on the given array
- Element -> Presents the element be executed now
- Index -> The index of the executed element right now
- Array -> On which array we are working
- This -> Value used as this when executed the callback function
This method returns (True => In case ALL elements satisfy the condition , False => In case One element DOES NOT satisfy the condition)
Example:

```
  let myArray = [10,20,30,40,50]
   console.log(myArray.every((e)=> e <= 50 )); // true because all elements <= 50
   console.log(myArray.every(function(e){
     return e > 50
   })); // False because 50 is Not bigger than 50
```

23. Spread Operator (...)

This concept used to spread the elements on any data type you want
Examples:

```
// Example 1
let name = "Rana"
console.log(...name); // R a n a
console.log([...name]); // ["R","a","n","a"]
// Example 2
let myArray = [1,2,3,3,2,4,2,1,5,3,6]
console.log([...new Set(myArray)]); // [1,2,3,4,5,6]
// Example 3 Concat
let myarr1 = [1,2,3,4]
let myarr2 = [5,6,7,8]
console.log([...myarr1,...myarr2]); // [1,2,3,4,5,6,7,8]
// Example 4 push
let myFriends = ["Rana", "Omar"]
let newFriends = ["Ahmed","Weal"]
let allFriends = myFriends.push(...newFriends)
console.log(allFriends); //["Rana", "Omar", "Ahmed", "Weal" ]
```

- This is the link of the challenge ()

24. Regular Expressions(https://regexr.com/)

- What are the usage of regular expressions ?

  - Regular expressions are used to create some specific pattern to check for example on something that matches that pattern or not , Here we will learn two things:
  1. How to create that pattern with a lot of rules
  2. How to check the matching of some variable and the written pattern (will mention it in the creating of RegEx) , So let's start

- **How to write a regular expression ?**

  1. Syntax:
  We declare a variable or Not (We can write the regular expression Directly) and then Between // we write our pattern Then we write the Modifiers (I will explain it in step 2)
  Example:
  let Regex = /here we write the expression/ here we write the flag Or modifier

  2. what are The modifiers or flags ?
  - They are like a simple condition, in which you tell the compiler about the strategy of searching
  Types of flags :
  i -> Case - insensitive (Search for this pattern in small or capital letters , it doesn't matter)
  g -> Global (Search in the whole string and return all matched values not the first one you met)
  m -> Multiline (In case the target string consist of more than one line , then search in all lines)

  3. How to check the matching between the target and the pattern ?
  We have 2 methods :
  1. target.match(RegEx) => This method returns an array of the matched strings between the target and the pattern
  In case No matching returns (Null)
  2. RegEx.Test(target) => This returns a Boolean value (True "In case the RegEx successfully passed the test of the target ", False "If Doesn't")

  Examples:
  ```
  let myString = "Hello Rana Ahmed , I love you rana";
  let Regex = / Rana/ig // this pattern says (I am searching about (all rana words)g (captial or small)i)
  console.log(myString.match(Regex)); // ["Rana","rana"]
  ```

  All of the previous points, are like an intro let's dive deep

- **Ranges :**
  - (X|Y) => X or Y
  - [0-9] => range from 0 to 9 (0,1,2,3,4,5,6,7,8,9) (Important note : 10 is (1,0))
  - [^0-9] => ANY THING **except** the rage form 0 to 9 **(^ consider as invert or the word except , and can be applied on numbers , letters and everything)**
  - [a-z] => The range of all small letters
  - [A-Z] => The range of all capital letters

- [abc] => This means we search for any a, b and c letters
- (abc) => This means we search for abc **(which is group of letters come in that sequence exactly)**
- [a-zA-Z0-9] => Means all letters (capital and small)+ range of numbers from 0-9

- **Character Classes:**
  - . => Matches any character except newline or any other line breaker
  - \w => Matches word characters which are **[a-z, A-Z, 0-9, and Underscore]**
  - \W => Matches NON word characters
  - \d => Matches digits form 0 to 9
  - \D => Matches Non-digit characters
  - \s => Matches white space character
  - \S => Matches Non-white space character
  - \b => Matches at the beginning or the end of a word (It does the two functions, but it mainly depends on the place of writing it {Ex: /\bRana/ This means beginning } {/ Rana\b/ This means end })
  - \B => Matches NOT at the beginning or the end of a word

- **Quantifiers**

  All of them are written after the character or the character class (ex: \w+)
  Using all the next quantifiers ,We can control the number of the characters|| Digits we want
  - n+ => one or more ex: \w+ -> one char or more
  - n* => Zero or more ex : \w* -> Zero char or more
  - n? => Zero or one ex : \w? -> Zero char or one
  - n{} => Number of ex : \w{2} -> 2 characters
  - n{x,y} => Range ex : \w{2,6} -> From 2 to 6 characters
  - n{x, } => At least X ex : \w{2, } -> At least 2 characters
  - $ => Ends with something (Important note : End with means write want to want first then $) Example /rana$/ this means ends with rana
  - ^ => Starts with something (Important note : starts with means write ^ first then write what you want) Example /^rana/ this means starts with rana
  - ?= => Followed by something (by default , write what you want first then (?=))
  - ?! => NOT Followed by something (by default , write what you want first then (?!))

- **Can we use regular expressions directly in methods like replace() and replaceAll() ?**
  - For sure we can, as we know those accepts 2 param (the old value , the new value)
    In the old value we can write the RegEx which will help us to apply advanced concepts

```
let testString = "I Love @, @ helps me to wake up "
 console.log(testString.replaceAll(/@/ig,"Coffee"));
```

- This is the link of the challenge (https://github.com/Rana5Ahmed/JavaScript_Challenges/blob/main/Regular%20Expression%20Challenge.js)

25. OOP (Object Orientated Programming)

→ **What is the meaning of OOP ?**
- OOP Is A Paradigm or Style Of Code
- OOP Use The Concept Of Object To Design A Computer Program
- Its Not New => Simula Is The First OOP Programming Language At 1960
- Some Languages Support OOP and Some Not
- Object Is A Package Contains Properties and Functions That Work Together To Produce Something in Your Application. Functions Here Called Methods

→ **Why OOP ?**
- Create a Large and Complex Software Architecture in Organized Ways.
- Create Reusable Objects To Use in Your Application Easily With Inheritance.
- Hide Certain Parts Of Code in Your Object With Encapsulation To Prevent Mess With The Code.

→ **What is the syntax of that paradigm "OOP"?**
- We have two syntax (Old and ECs6)

1. What is the meaning of constructor => A constructor is a special function used to create and initialize an object created with a class. It typically sets up the initial values of the object's properties when a new instance is created.

Old => Here we use the basic concept of function (Always remember that the Constructor is a function)
Let me Explain with code :

```
 // Best practice is to write the name of the construstor (Capatial)
  // userName, userAge, gradYear => Those are normally parameters as you know
  function Information(userName, userAge, gradYear){
    // Let's Explain here => 1. this : Referes to the creates instance (Object) from that
constructor
    // name, age, grad => are the properties in that created object and thier values will be
passed as arguments
    this.name = userName
    this.age = userAge
    this.grad = gradYear
  }
  // Using new, we created a new instance (نموذج او نسخة) from Information with the passed
data
  let user1 = new Information("Rana",24,2024)
  console.log(user1.name); // Rana
  console.log(user1.age); // 24
  console.log(user1.grad); // 2024

  console.log(user1 instanceof Information ); // True
  //  هل ال constructor الخاص ب user1  هو Information ?
  console.log(user1.constructor === Information); // Also true
```

2. New Way of writing (ECs6) => Here we use 2 keywords (Class, constructor)
Let's explain also with code :

```
  // The new way => added class word then constructor word
  // Note that the concept is the same
  class Information{
    constructor(userName, userAge, gradYear){
    this.name = userName
    this.age = userAge
    this.grad = gradYear
  }
  }
```

→ **How to deal with the properties and methods inside the class ?**
- We deal with them normally as anything we deal with before
Example :

```
   class Information{
      constructor(userName, userAge, gradYear){
      // Those Are properties for each instance object
      this.name = userName.length<3 ? console.log("Please Enter valid name"): userName  // if
      this.age = userAge || "Unknown Age" // Logical Or
      this.grad = gradYear<2015? console.log("We are so sorry, This intern is Not for you"):gradYear
      //But this is a Method for each instance object also
      this.logging = function(){
        return `Hello ${this.name}, You are ${this.age} and your grad year is ${this.grad}`
      }
    }
    outSide = function(){
      return("This is also a Method is shared via the prototype")
    }
    }
    let user1 = new Information("Rana",24,2024)
    console.log(user1.name); // Rana
    console.log(user1.age); // 24
    console.log(user1.grad); // 2024
    console.log(user1.logging()); // Hello Rana, You are 24 and your grad year is 2024
    console.log(user1.logging); //Native Code (Will return the code as it is )
    console.log(user1.outSide()); // This is also a Method is shared via the prototype
```

→ **How Can we update a property based on the user request?**

- Simply:

```
class Information{
   constructor(userName, userAge, gradYear){
    this.name = userName
    this.age = userAge
    this.grad = gradYear
  }
  // This is a short hand of defining the method
  update(newName){
    this.name = newName
  }
  }
  let user1 = new Information("Rana",24,2024)
  console.log(user1.name); // Rana
  user1.update("Ahmed")
  console.log(user1.name); // Ahmed
```

→ **What are the build-in constructors ?**

- They are conustrouctors exist in the language itself

Let's Explore Together:

```
let numOne = 15;
let numTwo = new Number(15);
console.log(typeof numOne); // Number
console.log(typeof numTwo); // Object (Constructors are objects )
console.log(numOne instanceof Number); // False
console.log(numTwo instanceof Number); // True => Because in the creation of numTwo it uses Number()
console.log(numOne.constructor === Number); // True => true because even though numOne is a primitive, its constructor is still Number
```

Super Important Note : JavaScript wraps it temporarily in a Number object when you access properties/methods.

```
console.log(numTwo.constructor === Number); // True
```

→ **What is the meaning of Class Static properties and method ?**

- Those are some properties and methods that related to the class itself Not the instance objects (Which means the instance objects from that class can NOT Access those properties and method)

- We made property or method static using (Static) keyword

Example :

```
class Information{
      // This is a static property
      static count = 10;
      static print() {
        console.log("Hello");
      }
      constructor(userName){
      this.name = userName
    }
    }
    let user1 = new Information("Rana")
    console.log(Information.count); // Accessable using the class name itself
    console.log(user1.count); // Undefined
    console.log(Information.print()); // Hello => Accessable using the class name itself
    console.log(user1.print()); // TypeError
```

→ **Class Inheritance**

- **What is the meaning of inheritance ?**

It is a behavior when the child take some characteristics from its father (Parent)

We also can do the same thing in OOP (let some class inherit some values from another class)

Let's explain how :

```
//Parent Class
 class Mainclass{
   constructor(userName,userAge,userId){
     this.name = userName
     this.age = userAge
     this.id = userId
```

```
    }
  }
  // Derived Class
  class Inheitclass extends Mainclass{
    // Super Important Note (Put the wanted inherited properties in the constructor also)
    constructor(userName,userAge,userId,subject,grade){
      super(userName,userAge,userId) // Here we avoid to write this. 3 times Again
      this.subject = subject
      this.grade = grade
    }
  }
  let user1 = new Mainclass("Rana",24,858)
  console.log(user1); //Mainclass {name: 'Rana', age: 24, id: 858}
  let user2 = new Inheitclass("Ahmed",28,145,"Math",98)  // The inherited properties is
correctly accessible in the class
  console.log(user2); //Inheitclass {name: 'Ahmed', age: 28, id: 145, subject: 'Math',
grade: 98}
```

→ **Class Encapsulation :**

What is the meaning of Encapsulation ?

- Is to make some properties and methods private , Class Fields Are Public By Default.

Why Encapsulation ?

- Guards The Data Against Illegal Access.
- Helps To Achieve The Target Without Revealing Its Complex Details.
- Will Reduce Human Errors.
- Make The App More Flexible And Manageable.
- Simplifies The App.

How to apply Encapsulation ?

Let's explain using code :

```
//Parent Class
  class Mainclass{
    // 1. First we have to define the private property
    #id
    constructor(userName,userAge,userId){
      this.name = userName
      this.age = userAge
      // 2. when you write this sentence put also #
      this.#id = userId

    }
    // important note : Private identifiers can Only be used in the class body
    checkId(){
      // For example check this private field
        return this.#id>1000? console.error("The id is Not correct"): this.#id
      }
      }
      let user = new Mainclass("rana",24,215)
      console.log(user.checkId());
```

Extra Example (Inheritance + encapsulation)

```
// Parent Class
class Mainclass {
  #id         // Private field '#id' declared ONLY in Mainclass, not accessible outside this class
  constructor(userName, userAge, userId) {
    this.name = userName
    this.age = userAge
    this.#id = userId   // Initialize private '#id' inside parent class only
  }
  checkId() {
    // Method accesses private '#id' within its own class
```

```
    // If id > 1000, log error, else return the id
    return this.#id > 1000
      ? console.error("The id is Not correct")
      : this.#id
  }
}
// Child Class inheriting from Mainclass
class inhertance extends Mainclass {
  constructor(userId, gender) {
    super(undefined, undefined, userId)  // Call parent constructor with undefined name, age, but pass id
    this.gender = gender
  }
  getId() {
    // This causes an error:
    // ReferenceError: Private field '#id' must be declared in an enclosing class
    // Because '#id' is private to Mainclass, NOT inherited or accessible here
    return this.#id
  }
}
let user = new inhertance(254, "Female")
console.log(user.getId());
// Throws ReferenceError: Cannot access private field '#id' outside class 'Mainclass'
// Because '#id' does NOT exist in 'inhertance', and private fields are not inherited.
console.log(user.checkId());
// Output: 254
// Works because 'checkId()' is declared in Mainclass and can access its private '#id'
```

→ **Prototype**

- What is the meaning of Prototype?

They are the mechanism by which JavaScript objects inherit features from one another.

Let's explain More :

When we declare first some class, then create an object from it , hence we find that object (Inherit) has all the properties and methods which were inside the class, right ?

Example :

```
class Data {
      constructor(prouduct, color, releasedYear){
        this.prouduct = prouduct
        this.color = color
        this.releasedYear = releasedYear
      }
    }
    let prouduct1 = new Data("car","red",2021)
    // Here the objected which is created from the class normally access all properties inside it
    console.log(prouduct1.prouduct, prouduct1.color, prouduct1.releasedYear);
```

This happened due to the concept of prototype , Note that: in the prototype list: the constructor of product1 is (Class Data) and the prototype is object. If you open the prototype of the class Data we will find that the prototype is (the object constructor)

→ **How to add some property or method to our prototype ?**

• We use .prototype is used to add methods or properties to the class itself, so that all instances created from it can access them.

Let's Explain more using code:

```
class Data {
      constructor(prouduct, color, releasedYear){
        this.prouduct = prouduct
        this.color = color
        this.releasedYear = releasedYear
```

```
    }
}
let prouduct1 = new Data("car","red",2021)
```

Super important note => We can't add method or property using .prototype to the product1 object

```
// add property in the Data prototype
Data.prototype.company = "BMW"
console.log(prouduct1.company); // BMW normally accessible

// add property in the Object constructor prototype
Object.prototype.newData = "This is coming from Object constructor"
console.log(Data.newData); //  "This is coming from Object constructor" => Normally accessible in Data class
console.log(prouduct1.newData); //  "This is coming from Object constructor" => Normally accessible in product1 object
```

→ **Object Meta Data And Descriptor**

- Here we will learn some ways to add or edit properties and method in external way, which is outside the whole object

- How we can do that (Add or update one property at time) ?

We use Object.defineProperty(the target Object,property_key, Descriptor)

let's Explain more using code:

```
let letters = {
  a: 1,
  b: 2
};
Object.defineProperty(letters, "c", {
  writable: true,
  enumerable: true,
  configurable: true,
  value: 3
});
console.log(letters); // {a: 1, b: 2, c: 3}
```

Explanation:

1. Object => The built-in global object (not a constructor in this case).
2. .defineProperty => A method used to add or update a property on an object. It takes 3 parameters:
   1. The target object we want to operate on.
   2. The property name (as a string).
   3. An object that contains property descriptors:

- writable:

If true → You can edit the value of the property.
If not provided → Defaults to false (you cannot change the value).

- enumerable:

If true → The property will appear during object iteration (e.g. for...in, **Object.keys**).
If not provided → Defaults to false (the property will be hidden from loops).

- configurable:

If true → You can delete the property (e.g., delete letters.c) or redefine it using defineProperty again.
If not provided → Defaults to false (you cannot delete or reconfigure the property).

- value:

The value you want to assign to the property.

- How we can do that (Add or update Multiple properties at the same time) ?

We use **Object.defineProperties(Object , {property:{Descriptors}})**

Note that : The concepts of Descriptor are the same

Let's talk an example using the new format:

```
let letters = {
  a: 1,
  b: 2
};
Object.defineProperties(letters,{
```

```
    c:{
    writable: true,
    enumerable: true,
    configurable: true,
    value: 3
    },
    d:{
    configurable: true,
    value: 4
    },

    e:{
    enumerable: true,
    value: 5
    },
  });
  console.log(letters);  //{a: 1, b: 2, c: 3, d: 4, e: 5}
```

→ How to know the descriptors of each property or of the whole object ?
- For each property we use `Object.getOwnPropertyDescriptor(target_Object,target_Property)`

```
Example :
console.log(Object.getOwnPropertyDescriptor(letters,"a")) // all of the descriptors are true (We defined it
using the normal way);
console.log(Object.getOwnPropertyDescriptor(letters,"d")) //  writable: false ,enumerable: false,
configurable: true;
```

- For the whole object we use `Object.getOwnPropertyDescriptors(target_Object)`

```
console.log(Object.getOwnPropertyDescriptors(letters)); // Give us all descriptors for all properties
```

26. Date And Time

→ We have two ways to express the date and time :
1. Date constructor => Which gives you the current date and time in that moment exactly
2. Date.now () => This gives us the time from Epoch Time Or Unix Time In Computer Science which Is The Number of Seconds Since January 1, 1970 To the current Second and it gives you this value in Milliseconds

Let's talk examples on both of them:

```
   let currentTime = new Date()
    console.log(currentTime); // Fri May 23 2025 20:56:37 GMT+0300 (Eastern European Summer Time) "Which is
the current time"

    let epochTime = Date.now()
    console.log(epochTime); // 1748023029171 "Which are the number of ms from 1/1/1970 till now"
    console.log(epochTime/1000/60/60/24/365); // 55.42 years "Which are the number of years from 1/1/1970
till now"
```

→ **How to get the date and the time ?**
- We have several methods to get those values exactly , Let's talk about them:

1. getTime() => This will return the time from the Unix epoch till the current millisecond (The returned value will be in ms)

```
   let currentTime = new Date() // This line will be with us in all experiments
    console.log(currentTime.getTime()); // 1748023672126
```

2. getDate() => This method will return the day of the current month (Here we are dealing normally with current time)

```
   console.log(currentTime.getDate()); // Returns the day-of-the-month (23)
```

3. getFullYear() => Will get the full year (بس كدة ؟ اومال انت فاكر اية)

```
   console.log(currentTime.getFullYear()); // Returns the year (2025)
```

4. getMonth() => This will return the month as a number (Super important note => Here it starts from Zero-based "Which mean January is index 0 and so on")

```
   console.log(currentTime.getMonth()); // Returns the index of the month (4)
```

5. getDay()=> Returns the index of the day of the week (Zero-based "starting from Sunday")

```
   console.log(currentTime.getDay()); // Returns the index of the day within the week (5)
```

6. getHours() => Will return the current hour

```
console.log(currentTime.getHours()); // Returns the hours in the date (21)
```

7.getMinutes() => will return the current minutes

```
console.log(currentTime.getMinutes()); // Returns the minutes in the date (25)
```

8.getSeconds() => will return the current seconds

```
console.log(currentTime.getSeconds()); // Returns the seconds in the date (15)
```

→ **How to set the date and time ?**

- we have also several methods to change the current date and time, let's talk about them:

1. setTime() => Here you put Milliseconds which descripes the time passed after 1/1/1970

```
currentTime.setTime(0) // No time passed
console.log(currentTime); // Thu Jan 01 1970 02:00:00 GMT+0200 (02:00:00 due to my time zone)

currentTime.setTime(1500000) // Note that this number is in ms
console.log(currentTime); // Thu Jan 01 1970 02:25:00 (02:00:00 due to my time zone)
```

2. setDate() => This method sets the day of the current month (+ve > the number of the days in that month we will move to the days of the following month , -ve => Then we will go back to the previous month)

```
// Note that the current date is 23/5/2025
currentTime.setDate(15) // Change the day to 15
console.log(currentTime); // Thu May 15 2025 21:58:12

currentTime.setDate(35) // Move 4 days to the next month
console.log(currentTime); // Wed Jun 04 2025 21:58:59

currentTime.setDate(-5) // Move 5 days to the previous month
console.log(currentTime); // Fri Apr 25 2025 22:00:18
```

3. setFullYear(year, month => Optional [0-11], day => Optional [1-31]) => set the year to whatever you want (Don't forget that the month is Zero-Indexed)

```
currentTime.setFullYear(2001,9,5) // Month and day are optional
console.log(currentTime); // Fri Oct 05 2001 22:06:09 GMT+0200 "Ranon's Birthday"
```

4. setMonth(month => [0-11], day => Optional [1-31]) => set the month to whatever you want

```
currentTime.setMonth(9,5) // day here is Optional
console.log(currentTime); // Sun Oct 05 2025 22:07:57 GMT+0300
```

5.setHours(Hours [0-23], Minutes => Optional [0-59], Seconds => Optional [0-59], MS => Optional [0-999])=> Here we set the hours, min, and sec

In case min >60 the hours will increase by 1 and also in sec >60 the min will increase by 1

```
currentTime.setHours(15,80,80) // day here is Optional
console.log(currentTime); // Fri May 23 2025 16:21:20 GMT+0300
```

6.We have also setMinutes(), and setSeconds()

→ **How to know the execution time taken in some operation?**

- Example with code (using what we took):

```
let startTime = new Date()
for(let i =0; i<1000; i++){
  let div = document.createElement("div");
  let divText = document.createTextNode(`${i}`)
  div.appendChild(divText)
  document.body.appendChild(div)
}
let endTime = new Date()
let durationTime = endTime - startTime
console.log(durationTime); // Time of execution in ms
```

- Example using another method: (preformance.now())

Much more accurate than Date.now() (which only gives millisecond precision).

Returns a **floating point number** like 1034.2346, meaning **1034 milliseconds and 234.6 microseconds** since the page started loading.

Not affected by changes to the system clock (unlike Date).

```
let start = performance.now()
for(let i =0; i<1000; i++){
  let div = document.createElement("div");
  let divText = document.createTextNode(`${i}`)
  div.appendChild(divText)
  document.body.appendChild(div)
}
let end = performance.now()
let durationTime = end - start
console.log(durationTime); // Time in ms also
```

- Example using another method: (preformance.mark())
Creates a named timestamp (a "mark") in the browser's performance entry buffer. You can later use it with performance.measure() to calculate durations between marks.

```
performance.mark("Start")
  for(let i =0; i<1000; i++){
    let div = document.createElement("div");
    let divText = document.createTextNode(`${i}`)
    div.appendChild(divText)
    document.body.appendChild(div)
  }
  performance.mark("End")
  performance.measure("Execution time","Start","End")
  const measure = performance.getEntriesByName('Execution time')[0];
  console.log(`${measure.name} took ${measure.duration} ms`); // Execution time took 2.799999952316284 ms
```

performance.mark("Start")
This sets a performance mark named "Start" right before the code block starts. Think of it as starting a stopwatch.

performance.mark("End")
This sets another performance mark named "End" after the loop finishes. This is like stopping the stopwatch.

performance.measure("Execution time", "Start", "End")
This creates a named measurement called "Execution time" using the time between the two marks ("Start" and "End"). It calculates how long the loop took to run.

const measure = performance.getEntriesByName('Execution time')[0];
This retrieves the first measurement object with the name "Execution time" from the browser's performance entry list

27. Generator Functions

→ **What is the meaning of generator function?**

- It is a function that executed whenever you want (و اية الجديد؟). This function has 2 special things : 1. It returns an object , 2. It is iterable (بمعنى انك لما بتعمل كول للفانكشن اول سطر بيتنفذ او بمعنى اصح بيخرج من خط "الانتاج" و ساعتها بتروح للسطر اللي بعده علشان تنفذه)
Let's see example with code:

```
// We must put * before the function's name to define this is a generator function
function *generate(){
      yield 1
      console.log("The log between yield 1 and yield 2");
      yield 2
      console.log("The log between yield 2 and yield 3");
      yield 3
    }
    console.log(generate); // Will give you native code
    let funcVariable = generate()
    // How ta access each yield ? => By using .next() such as in set()
    console.log(funcVariable.next()); // This line will give us yield 1 => {value: 1, done: false}
"done:false => Because the function has more yield yet"
```

Super important Note => The yield stops all other execution till it ends: Here the log statement will not displayed until you call the next yield

```
    console.log(funcVariable.next()); // This will give you 2 things : 1- The log between yield 1 and yield
```

```
2 , 2- {value: 2, done: false}
    console.log(funcVariable.next()); // This will give you 2 things : 1- The log between yield 2 and yield
3 , 2- {value: 3, done: false}
    console.log(funcVariable.next()); // {value: undefined, done: true}
```

→ Can I make a generator function Delegate another generator function (دالة تنوب عن دالة تانية)?
- Yes we can, let's see an example with code, In the following code we call genrator function inside another generator function.

```
    function *generateNumbers(){
      yield 1
      yield 2
      yield 3
    }
    function *generateLetters (){
      yield "a"
      yield "b"
      yield "c"
    }
    function *all(){
      // Because the yield is another generator function we must put * in the yield
      yield* generateNumbers()
      yield* generateLetters()
      yield* [4,5,6] // Here we put the * to return each element in the array separated
    }
    let allVariable = all()
    console.log(allVariable.next()); // This will give us the first yield in the first called function
(generateNumbers()) => which will give us {value:1,done:false}
    console.log(allVariable.next()); //{value:2, done:false}
    console.log(allVariable.next()); //{value:3, done:false}
    console.log(allVariable.next()); //{value:a, done:false}
    console.log(allVariable.next()); //{value:b, done:false}
    console.log(allVariable.next()); //{value:c, done:false}
    console.log(allVariable.next()); //{value:4, done:false}
    console.log(allVariable.next()); //{value:5, done:false}
    console.log(allVariable.next()); //{value:6, done:false}
    console.log(allVariable.next()); //{value: Undefined, done:true}
```

→ **Can I stop at specific yield and don't continue the function ?**
- Yes we can, Using return() => This will stop all yields after it

1. If you used return() outside the function them all the .next() after it will be {value: undefined , done: true}

Let's see an example:

```
 function *generateNumbers(){
      yield 1
      yield 2
      yield 3
    }
    let numbers = generateNumbers()
    console.log(numbers.next()); //{value:1,done:false}
    console.log(numbers.return()); //{value:undefined, done:true} => We stop all the following yields
    console.log(numbers.next()); //{value:undefined, done:true}
    console.log(numbers.next()); //{value:undefined, done:true}
```

2. What if we use it inside the function, The same concept will be applied (all yields after the return will be unreachable)

```
    function *generateNumbers(){
      yield 1
      yield 2
      return
      yield 3 // Unreachable code
    }
    let numbers = generateNumbers()
    console.log(numbers.next()); //{value:1, done:false}
    console.log(numbers.next()); //{value:2, done:false}
    console.log(numbers.next()); //{value:undefined, done:true} // The third yield is unreachable due to the
usage of return
    console.log(numbers.next()); //{value:undefined, done:true}
```

→ Let's Generate infinite sequence of numbers (It is super simple):

Code:

```
   function *infinteNumbers(){
      let counter = 1
```

```
  while (true) {
    yield counter++
  }
}
let infinitNum = infinteNumbers()
console.log(infinitNum.next().value); // 1
console.log(infinitNum.next().value); // 2
console.log(infinitNum.next().value); // 3
console.log(infinitNum.next().value); // 4
console.log(infinitNum.next().value); // 5
console.log(infinitNum.next().value); // 6
console.log(infinitNum.next().value); // 7
console.log(infinitNum.next().value); // 8
console.log(infinitNum.next().value); // 9
console.log(infinitNum.next().value); // 10
console.log(infinitNum.next().value); // 11
console.log(infinitNum.next().value); // 12
console.log(infinitNum.next().value); // 13
console.log(infinitNum.next().value); // 14
console.log(infinitNum.next().value); // 15
```

28. Modules

- **What is the meaning of module ?**

- Modules are files that contains some variables, functions, objects we need them in our main file, so we have 2 scenarios: 1. you need to Import something from another file you created to other file, 2. Import from built-in modules in language
In the first scenario you have to export "in the module file " and import "in your file"
Let's take an example:
Here we will have 3 files
1. HTML file

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <title>Learn JavaScript</title>
  </head>
  <body>
    // Super Important note the removing of type ="Module" will cause an error => Uncaught SyntaxError
    <script src="module.js" type="module"></script>
    <script src="app.js" type="module"></script>
  </body>
</html>
```

2. Module File
This is the module file
Here we have 2 ways to export those data
1. write export before the declaration like the following:

```
export let name = "Rana"

let obj ={
    age: 24,
    state: "Single"
}
let arr = [2001,2025]
function sayHello(){
    return "Hello"
}
```

2. Export all data in one line at the end of file using export and {}

```
export{obj,arr,sayHello}
```

3. App File
Here we will Import the data we need
And we have ONLY 1 method to do that
Super Important note => Don't forget to write the path correctly
Here also we can rename them as we want

```
import { name,obj,arr,sayHello as s } from "./module.js";
```

```
// All the values are normally accessible
console.log(name); // Rana
console.log(obj); // {age: 24,state: "Single"}
console.log(arr); // [2001,2025]
console.log(s()); // Hello
```

What if we need to import all the module in your file ?
Super easy:

```
import* as all from "./module.js";
console.log(all.name); // Rana
console.log(all.obj); // {age: 24,state: "Single"}
console.log(all.arr); // [2001,2025]
console.log(all.sayHello()); // Hello
```

- **What is the meaning of default export ?**
  - It is a unique option within the whole file (We can have ONLY one default export)
  Let's talk an example :
  Module file (We will change that small part only)

```
export default function sayHello(){
  return "Hello"
  }
```

App file:

Super important notes:

1. You can name the default export as you want (hi here is a new name of sayHello function)
2. The default export must be imported outside the {}

```
import hi,{name,obj,arr}from "./module.js";
console.log(name); // Rana
console.log(obj); // {age: 24,state: "Single"}
console.log(arr); // [2001,2025]
console.log(hi()); // Hello
```

29. JSON "JavaScript Object Notion"

➢ What is JSON?

JSON is Derived From JavaScript and Formatted For Sharing Data Between Server (back-end) And Client (Front-end).
Now we use Json as an Alternative To XML.
File Extension Is .json

➢ Why JSON?

- Easy To Use And Read compared to XML (which uses tags in his structure) , if you read a comparison between both of them read that article (https://www.geeksforgeeks.org/difference-between-json-and-xml/)
- Used By Most Programming Languages And Its Frameworks
- You Can Convert JSON Object To JS Object And Vice Versa (This way is super easy and we will learn how to do that)

➢ JSON syntax:

1. Data Added Inside Curly Braces { }
2. Data Added With Key : Value
3. **Key Should Be String Wrapped In Double Quotes**
4. Data Separated By Comma
5. Square Brackets [] For Arrays
6. Curly Braces {} For Objects

➢ Available Data Types

- String, Number, Object, Array, Boolean Values, null

Super Important notes:

1. In JSON file there is no function or any operation that made
Always remember => We share data between the two sides (server-client)
2. JSON must have **only one root object or array** , we can say JSON file is a one large object or array
3. Comments are not permitted in JSON.

Let's take an example with code:

Now we created a new file with .json extension (Focus on the syntax "It is quite similar to JS objects ")

```
{
  "name":"Rana",
  "age":24,
  "favNumbers":[3,5,10,12],
  "hoppy":{
    "cook":"pasta",
    "music":"sad"
  },
  "boolean": true,
  "nullValue": null
}
```

How to convert JSON code to JS and verse vise?

- we have 2 method:

1. `JSON.parse()` => Used to convert the JSON "String" to an JS object
2. `JSON.stringify()` => Used to convert the JS Object to a JSON string

Let's talk an example with code (in that example we will deal with JSON data in our main js file)

```
//we are in app.js , and this is a JSON data
let jsonData = '{"name":"Rana","Age":24}' // Note here we used '' to avoid the escaping of ""
console.log(jsonData); // {"name":"Rana","Age":24}
console.log(typeof jsonData); // String

// here we will convert that string into an object
let jsObject = JSON.parse(jsonData)
console.log(jsObject);
/*{name: 'Rana', Age: 24}
Age: 24
name: "Rana"
[[Prototype]]: Object */
console.log( typeof jsObject); // Object

// Let's do the opposite operation (From jsObject TO JSON string)
let jsonDataAgain = JSON.stringify(jsObject)
console.log(jsonDataAgain); // {"name":"Rana","Age":24}
console.log(typeof jsonDataAgain); // String
```

What is the meaning of API (Application Programming Interface)?

An API is a set of rules, protocols, and tools that allows one software application to interact or communicate with another.

• Think of it as a middleman or a messenger that:

1. Takes a request from one system
2.Passes it to another system
3. Then returns the response back.

- API simulation in the Real life:

Imagine you're in a restaurant:

- The **menu** is the API. It tells you what you can ask for.
- You (the client) place an order.
- The **waiter** (API) takes your request to the kitchen (the server).
- The kitchen prepares the food (the data).
- The waiter brings it back to you.

- **Here some API for your GitHub**

1. https://api.github.com/users/Rana5Ahmed
2. https://api.github.com/users/Rana5Ahmed/repos

- Here also a great website that related to JSON format (https://jsonformatter.org/0d1446#)

30. Asynchronous vs Synchronous Programming

Synchronous :

- ***Operations*** is synchronous programming Runs in Sequence (each operation waits until the previous operation finished its work) , In another meaning Each Operation Must Wait For The Previous One To Complete

- ***Story From Real Life*** => The line in any ticket machine , you will never get you ticket till your turn (So, you have to wait all people in the line before you)

- ***Example using code:***

```
    console.log(1);
    console.log(2);
    alert("Wait")
    console.log(3); // This will wait until all the other operations be executed (When you click ok for the alert then 3 will be logged )
```

➢ Asynchronous:

- ***Operations*** Runs In Parallel which Means That An Operation Can Occur while Another One Is Still Being Processed
- ***Story From Real Life*** => You are in a restaurant and you want to order food , Is it logical to wait the next table to order too ?! For sure No , So the orders in the restaurant is following the concept of Asynchronous (Parallelism) Two or more operations are executed at the same time
- ***Example Using code:***

```
   console.log(1);
    console.log(2);
    setTimeout(() => console.log("Hello after 2s"), 2000); // This will Normally executed in the background and after 2s will logged "Hello after 2s"
    console.log(3); // This will not wait the setTimeOut (Will Normally executed after 1,2)
    /* So the output In console will be :
    1
    2
    3
    Hello after 2s
    */
```

31. Call Stack and Web API

→ JavaScript Engine Uses A Call Stack To Manage Execution Contexts

→ **How It works (What is Mechanism To Make The Interpreter Track Your Calls) ?**

1. **When Function Called It Added To The Stack** which Works Using LIFO Principle => Last In First Out
2. Code Execution Is Synchronous.
3. When Function Executed It Removed From The Stack
4. After Function Is Finished Executing The Interpreter Continue From The Last Point

→ **What is the Web API?**

Methods Available From The Environment => Browser

→ **What about handling between the stack and the web API?**

Call Stack Detect Web API Methods And Leave It To The Browser To Handle It
Let's talk a piece of code to understand more:

```
   setTimeout(() => console.log("Web API"), 0);
    function one (){
    console.log("One");
    }
    function two (){
        one();
        console.log("Two");
    }
    function three (){
        two()
        console.log("three");
    }
    three(); // Here three is added to stack
    // The web API will wait until the stack become empty and then start its work and log Web API
    // So the output is => One , Two, Three, Web API
    /* Stack Simulation
================================
 return "One"
================================
 function one (){
     console.log("One");
    }
================================
  function two (){
    one();
    console.log("Two");
```

```
    }
================================
 function three (){
    two()
    console.log("Three");
    }
================================
*/
```

32. Event Loop and callback hell

1. First let's establish some concepts:

→ JavaScript Is A Single Threaded Language "All Operations Executed in ONE Thread"
→ Call Stack Track All Calls and based on FILO Every Function Is Done Its Popped Out of the stack
→ And as we said When You Call Asynchronous Function It Sent To Browser API (He deals with it)
→ Asynchronous Function Like Settimeout Start Its Own Thread **(This is a theoretically)** **في الواقع زي ما قولنا هي بتتكون من ثريد واحدة بس**
→ Browser API ***Act As*** A Second Thread, Browser API Add The Callback To Callback Queue "Someplace to wait in 'Waiting Room' "
→ Event Loop Wait For Call Stack To Be Empty
→ Event Loop Get Callback From Callback Queue And Add It To Call Stack (This happened when all the synchronous operations be executed)
→ But here, The callbacks will be executed based on the concept of queue, Callback Queue Follow FIFO "First In First Out" Rule
→ Super Important Note :
- In the web API , There is some functions and methods that is synchronous and other ones are Asynchronous
ex:(Console.log() => Synch, setTimeOut() => Async), The whole concept of callback queue is applied on the Async Operations
Example with code:

```
// First, One scan is made to decide which operation is sync and which is async
    // Here the First setTimeOut() Will be inserted in the callback queue(The stack is not empty)
    // Here the Second setTimeOut() Will be inserted also in the callback queue(The stack is not empty)
        setTimeout(() => {
            console.log("Five"); //5  //first element in the queue "FIFO" will now be logged
after  console.log("Four");
        }, 0);
        setTimeout(() => {
            console.log("Six");//6 // Second element is the queue will now be logged
after    console.log("Five");
        }, 0);
        console.log("One"); // sync => One  1
        console.log("Two"); // sync => Two  2
        console.log("Three"); // sync => Three 3
        console.log("Four"); // sync => Four "Stack is empty now"  4
```

33. Asynchronous JavaScript And XML (Ajax)

→ AJAX is an approach To Use Many Technologies Together [HTML, CSS, Js, DOM]
→ It Use "XMLHttpRequest" Object To Interact With The Server
→ You Can Fetch Data Or Send Data Without Page Refresh
Let's talk an example to see what that means:

```
    let interactWithServer = new XMLHttpRequest()
    console.log(interactWithServer); // This is an object contains several items we will talk about
```

Sample on the Things inside the XMLhttpRequest() object:
1. ReadyState => This describe the state of the request, we have 5 states represented with numbers from 0-4
[0] Request Not Initialized
[1] Server Connection Established
[2] Request Received
[3] Processing Request
[4] Request Is Finished And Response Is Ready
2.Status => Represent the state of the Response, This has multiple values (search about them) but the most common values are:
[200] Response Is Successful
[404] Not Found

→ **Let's deal with a Real API**
CODE: Super Important

```
    let interactWithServer = new XMLHttpRequest()
```

```
    interactWithServer.open("Get","https://api.github.com/users/Rana5Ahmed/repos")
    interactWithServer.send()
    console.log(interactWithServer); // Here we find readystate:4 , status: 200
    // Now we want to log the data
 Super Important Note :
 Arrow functions do not bind their own this; instead, they inherit this from the surrounding lexical context
(outside the function)
    // So If you want to use this use the regular function()
    interactWithServer.onreadystatechange = function(){
        // First we have to make sure the request and the response both are successfully executed
        if(this.readyState ===4 && this.status===200){
            // Here we convert the string into an object to iterate on that object
            let jsObject = JSON.parse(this.responseText)
            for(let i = 0; i< jsObject.length;i++){
                let div = document.createElement("div")
                let textNode = document.createTextNode(jsObject[i].name) // The content will be the repo
name
                div.style.cssText="padding:5px; color:navy" // Just to remember the styling
                div.appendChild(textNode)
                document.body.appendChild(div)
            }
        }
    }
```

34. Callback hell OR Pyramid Of Doom

1. **What is the meaning of callback hell ?**
   - First, we can define what is the meaning of callback function, A Function That Is Passed Into Another One As An Argument To Be Executed Later

2. **What happened if function call another function and that function calls another one and so on ?**
   - Callback hell will happen , This image will explain the meaning more:

3. 

```
// Callback Hell


a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});

```

4. Example with code:

```
setTimeout(() => {
    //This is a callback function (function takes another function as a parameter)
    console.log("CB1");
    // The callback takes another Callback
    setTimeout(() => {
        console.log("CB2");
        // callback inside callback inside callback
        setTimeout(() => {
        console.log("CB3");
        // callback inside callback inside callback inside callback "Dud!!, what are you doing!!"
        setTimeout(() => {
        console.log("CB4");
                }, 1000);
            }, 1000);
    }, 1000);
}, 1000);
```

35. Promise

→ **What is the meaning of promise?**
- Promise In JavaScript Is Like Promise In Real Life, you Promise Is Something Will Happen In The Future

→ **What is the purpose of Promise?**

- **Promise Is The Object That Represent The Status Of An Asynchronous Operation And Its Resulting Value**
- They Avoid Callback Hell

→ As we said the Promise represents the status (Remember status => Targets Response) of the operation, **so How many Status promise has ?**

- Pending: Initial State
- Fulfilled: Completed Successfully
- Rejected: Failed

→ Story

- Once A Promise Has Been Called, It Will Start In A Pending State
- The Created Promise Will Eventually End In A Resolved State Or In A Rejected State
- Based on that state (Resolve || Reject) some callback function will be executed
- Just take this as an information in that time: Then => Takes 2 Optional Arguments [Callback For Success Or Failure]

→ Syntax :

Now let's talk about the syntax,

1. First we declare the promise (Which is Object) using new Promise() => new Promise accepts a function with 2 parameters each parameter is also a function

2. Promise (function(resolve, reject)), Both resolve and reject are functions "JS give them to you to control both states"

• resolve => Means that your promise is successed and there is the data or Value resolve(value), Value here represent the data, **super Important note: This value will be passed next to then**

• reject => Means that your promise is failed and there is the error and message resolve(value) Value here will represent the error, **super Important note: This value will be passed next to catch**

```
const myPromise = new Promise((resolve, reject)=>{
// This line is a simulation if you received the data or not, let's assume you are connected
let connect = true;
// then
if (connect) {
    // resolve function must now work, and the value is the data
    resolve("I got the data heeeee");
    // What if the connect = false "Failure"
} else {
    // reject function must work, with the value of error for example
    reject(Error("The connection Failed"));
}
})
```

Here we finished the promise, but we didn't do anything with that data or error

So, We have to use Then() to tell him "When the promise finishes, do one of these things

Then (***function that takes the resolve value and do whatever you want in*** , **function that takes the reject value and do whatever you want in**)

```
.then(
(resolveValue) => console.log(`${resolveValue}`), // I got the data heeeee
(rejectValue) => console.log(`${rejectValue}`) //Error: The connection Failed
);
```

→ Then, Catch and Finally:

- What is the difference between all of them?

• Then => This targets the success ONLY (If the promise is successed ONLY)

• Catch => This targets the failure ONLY (If the promise is Failed ONLY)

• Finally => This will be executed whatever the state of the promise (Success or Failed)

Let's Talk an example with code :

```
let myPromise = new Promise(function(resolve,reject){
      let myInformation ={
          name:"Rana",
          age:24,
          state:"S"
      }
      if (myInformation.state ==="single") {
          resolve(myInformation.state)
      }
      else {
          reject("Not Avaliable")
      }
  }).then(function(res){console.log(`The State is ${res}, Congrats`)
  }).catch(function(rej) {console.log(`${rej}, good luck in the next time`)})
```

```
    .finally(()=>console.log( "This line will be executed whatever the state of the promise")
)
```

Apply that scenario:

We Will Go To The Meeting, Promise Me That We Will Find The 4 Employees
.then(We Will Choose Two People)
.then(We Will Test Them Then Get One Of Them)
.catch(No One Came)
Code:

```
let myPromise = new Promise(function(resolve,reject){
let employee = ["Rana","Ahmed","Omar","Weal"]
    if (employee.length>0) {
        resolve(employee)
    }
    else {
        reject(Error("No One came"))
    }
}).then((res)=>
    {res.length = 2
    return res // Super important note : This will be returned to the next then
    }
)
.then((res)=> // Here the res which came from the previous then (employee. length now is 2)
{   res.length= 1
    console.log(res);
})
.catch((rej)=>console.log(rej))
.finally(()=>console.log( "This line will be executed whatever the state of the promise")
)
```

→ Promise and XHR

Let's talk code directly:

```
// Define a function called getData that takes an API URL as a parameter
let getData = function(Api) {
    // Return a new Promise to handle asynchronous behavior
    return new Promise((resolve, reject) => {
        // Create a new XMLHttpRequest object to send an HTTP request
        let dataRequest = new XMLHttpRequest();
        // Define what happens when the request loads
        dataRequest.onload = function() {
            // Check if the response was successful (status 200 and request fully done)
            if (this.status === 200 && this.readyState === 4) {

                // Parse the JSON response text and resolve the Promise with the data
                resolve(JSON.parse(this.responseText));
            } else {

                // If the response was not successful, reject the Promise with an error
                reject(console.error("No Data found"));
            }
        };
        // Set up the request to use the GET method and provide the API URL
        dataRequest.open("GET", Api);
        // Send the request
        dataRequest.send();
    });
};
// Call getData with a GitHub API URL and handle the result
getData("https://api.github.com/users/Rana5Ahmed/repos")
    // If the request is successful, log the name of the first repository
    .then((res) => console.log(res[0].name))
    // If there's an error, log the error message
    .catch((rej) => console.log(`Error ${rej}`));
```

→ **What if you want to collect number of promises and have a common destiny ?**

- We have to collect all promises in one array then apply the operation
- Then we use (All, All settled, Race)

• All() => All promises must be resolved to return all of them, in case one promise is rejected, then that collection will return the rejected value of the rejected promise ONLY
Let's talk an example:
Incase all Promises are resolved:

```
   let promise1 = new Promise((resolve,reject)=>{
        setTimeout(() => {
            resolve("This is from promise 1")
        }, 1000);
    })
    let promise2 = new Promise((resolve,reject)=>{
        setTimeout(() => {
            resolve("This is from promise 2")
        }, 1000);
    })
    let promise3 = new Promise((resolve,reject)=>{
        setTimeout(() => {
            resolve("This is from promise 3")
        }, 1000);
    })
    Promise.all([promise1,promise2,promise3])
    .then((res)=>{console.log(res);  // Array of 3 :(3) ['This is from promise 1', 'This is from promise 2',
'This is from promise 3']
    }, (rej)=>{console.log("Rejected:"+`${rej}`)})
```

In case we change promise3 into :

```
 let promise3 = new Promise((resolve,reject)=>{
        setTimeout(() => {
            reject("This is from promise 3 ")
        }, 1000);
    })
```

- Then the output became => Rejected:This is from promise 3 "Returns the rejected promise ONLY"

• All settled () => Takes an array of the promises and returns all the promises (resolved and rejected ones)
Same previous example put now we will use all settled
here will give us :

```
0: {status: 'fulfilled', value: 'This is from promise 1'}
1: {status: 'fulfilled', value: 'This is from promise 2'}
2: {status: 'rejected', reason: undefined}  // That come from promise 3
```

• Race() => The first promise finished (Whatever it ended with resolve or with reject)
Example with code:

```
   // This code will return  => This is from promise 1 (resolved)
    let promise1 = new Promise((resolve,reject)=>{
        setTimeout(() => {
            resolve("This is from promise 1")
        }, 1000);
    })
    let promise2 = new Promise((resolve,reject)=>{
        setTimeout(() => {
            resolve("This is from promise 2")
        }, 2000);
    })
    let promise3 = new Promise((resolve,reject)=>{
        setTimeout(() => {
            reject(console.error("Can't fetch data"))
        }, 3000);
    })
    Promise.race([promise1,promise2,promise3])
    .then((res)=>{console.log(res);
    },(rej)=>{console.log("Rejected:"+`${rej}`)})
```

Same Example with little change:

```
   // This code will return  => Rejected:Error: Can't fetch data (Rejected From Promise 3 due to shorter
waiting time)
    let promise1 = new Promise((resolve,reject)=>{
```

```
    setTimeout(() => {
        resolve("This is from promise 1")
    }, 1000);

})
let promise2 = new Promise((resolve,reject)=>{
    setTimeout(() => {
        resolve("This is from promise 2")
    }, 2000);

})
let promise3 = new Promise((resolve,reject)=>{
    setTimeout(() => {
        reject(Error("Can't fetch data"))
    }, 500);
})
Promise.race([promise1,promise2,promise3])
.then((res)=>{console.log(res);
},(rej)=>{console.log("Rejected:"+`${rej}`)})
```

36. Fetch API

- This way is a shorthand of the whole derma of the promises this Return A Representation Of the Entire HTTP Response
- fetch is a built-in JavaScript function used to make HTTP requests (like GET, POST, etc.) to servers — for example, to get data from an API. It replaces the older XMLHttpRequest with a cleaner, simpler, and more modern way to handle requests and promises

- **How it works**

1. fetch() returns a Promise.
2. If the request is successful, the Promise resolves with a Response object.
3. You usually call .json() or .text() on that response to get the actual data.
4. If there's a network error, the Promise rejects and goes to .catch().

Let's take example directly:

```
// We did the same operation but using fetch
fetch("https://api.github.com/users/Rana5Ahmed/repos")
.then((res)=>{
    return res.json()
})
.then((res)=>{
    console.log(res[0].name);
})
.catch((rej)=>{console.log(console.error(`${rej}Can't find the data`)
);
})
```

37. Async and Await

→ **What is the difference between the regular function and async function?**

- Async Before Function Mean This Function Return A Promise
- Async And Await Help In Creating Asynchronous Promise Behavior With Cleaner Style

Let's Take some example that we are doing the same thing using 2 different ways:

```
// Regular function that returns a promise:
 /*
 1. Here we have to write return new promise
 2. use res and rej in the ordinary way which increase the headache of the syntax
 */
 function returnedPromise(users){
     return new Promise((res,rej)=>{
         if (users.length>0){
             res(users)
         }
         else{
             rej(Error("No users found"))
         }
     })
 }
 returnedPromise([]).then((res)=> console.log(`The current users are ${res}`)
 ).catch((rej)=> console.log(rej)
 )
```

```
// Async Function
/*
1. Writing async is enough to tell you that function will return a promise
2. Here, we don't have to write res, rej and we write our syntax normally
*/
async function asyncFunction(users) {
    if(users.length>0){
        return users
    }
    else{
        throw new Error("No users found")
    }
}
asyncFunction(["Rana"]).then((res)=> console.log(`The current users are ${res}`)
).catch((rej)=>console.log(rej)
)
```

What is the benefit of Await?

- Await Works Only Inside Asnyc Functions
- Await Make JavaScript Wait For The Promise Result (which will change in the sequence a little bit)
- Await Is More Elegant Syntax Of Getting Promise Result

Example with and without await :

```
// In that structure the console will have
    /*
    Before promise
    After Promise
    This is the resolved of the promise
    */
    let myPromise = new Promise((res,rej)=>{
    setTimeout(() => {
        res ("This is the resolved of the promise")
        rej(Error, "This is the reject of the promise")
    },2000 );
    })
    function testPromise() {
        console.log("Before Promise");
        myPromise.then((res)=>(console.log(res)
        )).catch((rej)=>(console.log(rej)
        ))
        console.log("After promise");
    }
    testPromise()
```

- Using Async and Await :

```
    // In that structure (Async/Await) the console will have
    /*
    Before promise
    This is the resolved of the promise
    After Promise
    */
    let myPromise = new Promise((res,rej)=>{
    setTimeout(() => {
        res ("This is the resolved of the promise")
        rej(Error, "This is the reject of the promise")
    },2000 );
    })
    async function testPromise() {
        console.log("Before Promise");
       await myPromise.then((res)=>(console.log(res)
        )).catch((rej)=>(console.log(rej)
        ))
        console.log("After promise");
    }
    testPromise()
```