# JavaScript

JavaScript is Most commonly used programming language, which is used to develop web application, mobile application, desktop application, severs, game and etc. it is developed by Brendon Eich in 1995. It was called as , mocha >>> JavaScript. ES6 >> version supports all the browser.

# JavaScript features

* ***Dynamic*** *We are able to store the different values in the same datatype (var) // int, char >> everything is stored in var datatype.*
* ***Interpreted*** *Line by line execution of the code*
* ***Synchronous*** *It is similar to the Interpreted. but you can skip the line-by-line execution and we*

can execute the code when we need, but it’s possible by using asynchronous

* ***Platform Independent***
* ***Light weight Scripting language****>> No proper syntax is needed*
* ***Single Threaded*** *>> Can perform one action at a time*

# Browser and Name of the JavaScript Engine

Chrome >> V8 Edge >> Chakra

Firefox >> Spider Monkey

Safari >> JavaScript Core web-kit

# JavaScript Engine

JavaScript engine is a program which is responsible to compile and execute the JavaScript programs.

**Compiler** is a program which checks the syntax errors and converts high level language to machine level language.

**Interpreter** is the program which executes machine level language line by line.

**Syntax errors** are the errors which will occur at the time of compilation. These are also known as compile-time errors. Syntax errors can’t be handled.

Ex >> missing ; () " " etc...

**Exceptions** are the errors which will be occurred at the time of execution & also known as run-time errors. When we get exception, our execution flow will be disturbed. We can handle the exceptions by using try-catch block.

# Note:

1. *; are not mandatory in JavaScript but missing ; is not a good practice.*
2. *JavaScript is case sensitive.*

# Data Type

It decides which type of data to be stored in the memory.

# Types of Datatypes

* *Primitive*
* *Non- Primitive*

1. ***Primitive****: The data types which is provided by JavaScript is known as primitive Datatypes. Primitive Datatypes is also known as pre-defined datatypes. Below are the different types*
   * ***Number****: It stores any type of number. The range is 2^53 -1.*
   * ***String****: Any character that is stored in* ***" ", ' ',*** *or* ***` `*** *will be considered as a string.*
   * ***Boolean****: True (1) or False (0) value based on the scenarios.*
   * ***Null****: Empty value*
   * ***Undefined****: default value of JavaScript, declared but not initialized*
   * ***BigInt****: Number to be stored where the range is more than 2^53 -1*
   * ***Symbol*** *>> Acts as an Object but not frequently used*
2. ***Non-Primitive****: All the objects which are given by JavaScript (pre-defined objects) and the objects created by the developer is known as non-primitive datatypes.*

|  |  |  |
| --- | --- | --- |
| * ***Arrays*** | * ***Object*** | * ***Date*** |
| * ***Set*** | * ***Map*** | * ***Math*** |

# Output methods >>

* ***console.log ():*** *It’s a debugging tool and allows us to debug. Errors will be displayed on the console.*
* ***document.write():*** *Anything that needs to be printed on the webpage*
* ***document.writeln()****: Same as document.write() but leaves the space between the content.*
* ***Confirm****: It’s used to get the confirmation from the user. It has 2 buttons Ok and Cancel. If the user clicks Ok, Boolean true value is sent, On clicking Cancel, Boolean false value will be sent.*
* ***alert()****: It displays the warning messages and it has a button called OK*
* ***prompt()****: It accepts the input message from the user and it prints on the console page. It has text box with 2 buttons. OK and CANCEL. If the user clicks OK, entered value will be sent as a string (If no value is passed and clicked OK, then it displays nothing as default value is string. If it clicks on cancel then default value is null)*
* ***Dom related methods*** *>> In order to print the output from the console on the UI*

# Note>>

* *If i need to print any values from the confirm or prompt, it needs to be stored in a variable and*

I’ll need to print that variable

* *By default, the default the data is taken as a string*
* *Disadvantage if document.write() is that it over rides the UI of existing one and prints only the content of document.write()*
* *Never use anything apart from console.log*

# Syntax:

Keyword variable-name = value ;

# Keywords of JavaScript

###### Var

* ***Let***

###### Const

*var a****; // Declaration***

*a = 20****; // Initialization*** *a=40****;// Re-initialization*** *60+a* ***// Utilization***

|  |  |  |  |
| --- | --- | --- | --- |
|  | **var** | **let** | **const** |
| **Only declaration** |  |  |  |
| **Only initialization** |  |  |  |
| **Re- declaration** |  |  |  |
| **Re- initialization** |  |  |  |
| **Single line declaration and initialization** |  |  |  |
| **Single line re-declaration and re-initialization** |  |  |  |
| **Scope** | [Global scoped](https://www.geeksforgeeks.org/understanding-variable-scopes-in-javascript/#%3A~%3Atext%3Dtypes%20of%20scopes-%2CGlobal%20Scope%2C-%E2%80%93%20Scope%20outside%20the) or function scoped | only block scoped. | only block scoped. |
| **Hoisting** | Supports Hoisting | Doesn’t support Hoisting | Doesn’t support Hoisting |
| **Introduces** | JavaScript | ES6 | ES6 |

# Scope

Scope refers to the current context of code which determines the accessibility of variables to JavaScript

* ***Global scope:*** *Anything that is declared out the function and inside the program is global scope variable.*
* ***Local or Block scope:*** *If you declare anything in the block scope its in the block scope, but only var keyword can be accessed outside*

# JavaScript Operators

Operators are the symbols which performs operations on operands.

Syntax >> Expression = operand operator operand

# Types of Operators

|  |  |  |
| --- | --- | --- |
| * ***Arithmetic operators*** | * ***Unary operators*** | * ***Logical operators*** |
| * ***Assignment operators*** | * ***Relational operators*** | * ***Ternary operator*** |
|  | * ***Special Operator*** |  |

##### Arithmetic operators

* + *These are used to perform mathematical arithmetic operations.*

|  |  |
| --- | --- |
| ***+*** *2 +3 = 5* | ***/*** *4/2=2* |
| ***-*** *3-2=1* | ***%*** *>> 3/2=1* |
| ***\**** *2\*3 =6* | ***\*\**** *2\*\*3 = 8 (Power)* |

# Note:

* *Whenever we perform modules operation, if dividend is bigger than divisor it returns reminder. If divisor is bigger than dividend, it returns dividend.*

 + operator can be used for 2 operations. It acts as addition when the operation is performed between 2 numbers. If any operand is a string, it acts as concatenation.

*Example >> 2* ***+*** *2 = 4*

*"2"****+*** *3 = 23*

* *Whenever it is performing concatenation operation, the result will be a string*

##### Unary operators

* + *These are used to increment or decrement the values by 1.*
  + *There are 2 types of unary operators, Increment (++) and Decrement (--)*
  + *Increment operator increases the value by 1 and stores inside the same variable.*
  + *There are 2 types of increment and decrement operators*
    - *Pre-increment and Pre-decrement*
    - *Post increment operator and Post increment operator Example >>*

*++a --a a ++ a-- a+=5*

##### Logical Operators

* + *There are 3 types of Logical Operators.*
    - ***AND ( && ) :*** And *operator return true when all the condition are stratifies. Else it return false.*

*In And operator first condition is false is not going to check 2nd condition directly return false. *

* + - ***OR ( || ) :*** OR *operator return true if any one condition is stratifies.it return false when all the condition are false.*

*In OR operator first condition is true is not going to check 2nd condition directly return true.*

* + - ***NOT ( ! ):*** *It is used to negate the result. i.e, if the result is true, it returns false and if the result is false, returns TRUE*

##### Assignment Operators

= += -= \*= /= \*\*= %=

* + ***Note >> a+= n--> a= a= a+n***

1. ***Relational Operator***
   * *Relational operators are used to compare the values of 2 operands.*
   * *It returns Boolean value*
   * *If condition satisfies, it returns True else returns False*
   * *There are different types of relational operators, they are as follows*

##### Note >>

 **==** compares only values

 **===** compares both values and the data types, If any of it is different, then it returns false

> < >= <= != == === !==

##### Ternary Operator

* + *condition? True: false Ex>>*

***a=10***

***20>a? "good":"False" // Prints Good***

##### Special Operator

* + ***type of*** *>> It checks which type of data and returns the data type name in the form of String*
  + ***in***
  + ***of***
  + ***delete***

# Control Statements

Control statements are the statements which decides the flow of the execution of the program. Below are the 3 types of control statements.

* + **Conditional Statements**
    - **If statement:**

Whenever we use if statement, if the condition is true, the code inside the if-block is executed, then the code outside if block will be executed.

if the condition is false, code outside if-block will be executed. i.e, control will not get into if block.

Ex >>

let myAge = 25;

let standardAge = 18;

***Output :*** *Both the statements will be printed as the condition is true, after the if-block the statements will run as there is no else*

if (myAge >= standardAge)

{

console.log("Eligible to apply for DL")

}

console.log("Please dont apply now")

#### If else statements:

If the condition is true, it executes the statements in if loop and does not execute the statements in the else loop and continues with the code and vice-versa.

#### Ex >>

let myAge = 25;

let standardAge = 10;

**Output//** It Prints the value based on the condition. If the condition is true prints inside the if-loop else prints inside the else-loop

if (myAge >= standardAge)

{

console.log("Eligible to apply for DL")

}

else

{

console.log("Please dont apply now")

}

**Note >>** When we want to perform some tasks based on True or False then we use if else condition**.**

#### Nested if

If loop inside the if statement.

***Example****>>*

let dbEmail = [prajju@gmail.com](mailto:prajju@gmail.com)

let dbPassword = 007; let userEmail = prompt("Enter the email"); let userPassword = Number(prompt("Enter the Password")) if(dbEmail === userEmail ){

if(dbPassword === userPassword ){ console.log("Login Successful")

}

***Output*** *This checks if the dbemail with the entered email and if its true then again it checks for the dbpassword and the entered password which is already inside the if-loop*

else () {

console.log("Please re-enter the Password")

}}

else

{

console.log("User not found")

}

#### Programs based on the if loop.

**Program 1 >> Big and Small number**

**Program2 >> Even or Odd**

let a = 420. if (a%2 == 0)

{

console.log("The number is even")

}

else

console.log(The number is Odd)

let a = 10; b=4; if (a>b)

{

console.log("A is bigger")

}

else (

console.log("B is bigger")

}

#### Program3>> Divisible by 3

**Program4 >>81 is multiple of 3 or not**

*let a = 81*

*if (a%3==0)*

let a if(a%3==0)

{

console.log(The number is divisible by 3")

}

else

{

console.log("The number is not divisible by 3

}

#### Note::

**if elseif**

**>>***Multiple conditions to check then we use this*

#### Program1 >> Greater of 3 number

*let a, b, c;*

*if(a>b && a>c)*

*{*

*console.log("a is bigger")*

*}*

*else if (b>a && b>c)*

*{*

*console.log("b is bigger"0)*

*}*

*else*

*{*

*console.log("C is bigger")*

#### Switch>>

Its an alternative for if else-if ladder

#### Example >

*let day = "sunday" switch(day)*

*{*

*case "sunday" : clg("Weekend ends"); break; case "monday" : clg("Weekday Starts"); break; case "wednesday" :clg("middle of week"); break; case "friday" : clg("end of weekday"); break; case "saturday" : clg("Weekend starts ") default:clg("No Day is matching")*

*}*

#### Looping statements

* + - **For loop >>**
      * *It’s used to perform the same operations repeatedly*
      * *for loop will be used when we know the termination condition or when we know no. of iterations.*

###### Syntax >> for (initialization; condition; increment/decrement)

**Note** >> If the condition is false, it does not increment or decrement the value, it just exists the for loop

***Example >>***

let i,n;

for (i=0;i<n;i++)

{

console.log(i)

}

**O/P** >> Prints the value of i till it’s less than n

###### Program 1 >> odd numbers from 1 -10 from 10-1

let i; let i;

let n=10; let n = 1;

for (i=1; i<n;i++) for (i=10; i>=1; i--)

{ {

if (i%2 !==0) if (i % 2 !==0)

{ {

clg(i) clg(i)

} }

} }

#### Jump Statements

* + - **Break >>** *used to terminate the loops and also the switch statement.*
    - **Continue >>** *It skips the current line of execution and continues with the loop again*

***Example >>***

***Program 1 >> odd numbers from 1 -10 from 10-1***

|  |  |  |
| --- | --- | --- |
| let i; |  | let i; |
| let n=10; |  | let n = 1; |
| for (i=1; i<n;i++) |  | for (i=10; i>=1; i--) |
| { |  | { |
| if (i%2 !==0) |  | if (i==3 || 1==7) |
| { |  | { |
|  |  | **continue**; |
| clg(i) |  |  |
| } | } |  |
| if (i===7) |  |  |
| ***break;*** |  | console.log(i) |
| } | } |  |

###### While Loop

* + It is used when the no of iterations are not known
  + Entry control loop

#### Ex >>

let i=1; while (i<=10)

{

console.log(1) i++

}

* ***Do while >>***

Similar to a while but it’s called exit control loop

##### Programs

***Program 3 >> Multiples of 3 till 90***

***Program 4 >> find the factors of 90***

*for (i=1;i<=90;i++)*

*{ if(90%i===0)*

*{*

*console.log(i)*

*}*

*}*

for (i=0;i<=90;i++)

{ if(i%3===0)

{

console.log(i)

}

}

##### Program 5 >> Factorial of 9

*for (i=1;i<=n;i++){ fact = fact \* i;}*

##### program 6>> Given number is +ve or not

***Program 7 >> Even multiples of 5 till 200***

*let n =200;*

*for (i=0; i<=n; i+=5)*

*{*

*if ((n%i===0) && (i%2===0))*

*{*

*console.log(i)*

*}*

*}*

***Program 10 >> Sum of 10 natural numbers***

*let n, sum=0, i; for(i=0; i<=n; i++)*

*{*

*sum = sum+i;*

*}*

*console.log(i)*

***Program 11 >> Prime or not***

Count =0

for (i=2; i<=n/2;i++)

{ if (n%i===0) Count=1; Break;

}}

If (count===0) clg(Prime) else

clg(not Prime)

***Program 11 >> To Print prime***

Count = 0, I, number, n

For (number=1; number<=n; number++)

{

For(i=2; i<=n/2;i++)

{

If (number%i===0) Count ++;

Break;

}}

If (count===0 && number!=1) Clg(prime number is : number)

*let n;*

*if (n===0)*

*console.log("Enter some other number") while (n!==0)*

*{*

*if (n>0)*

*{*

*console.log("The number is positive")*

*}*

*else*

*{``*

*console.log("The number is negative")*

*}*

*}*

###### Program 8 >> Starting 3 factors of 50

*for (i=1;i<=50;i++)*

*{ if(50%i===0)*

*{*

*console.log(i) count++*

*if (count===3) break;*

*}*

*}*

##### Program 9 >> Prime factors of 120

*for (i=1; i<=120; i++)*

*{*

*count=0; if(120%i===0)*

*{*

*for (j=2; j<=1; j++)*

*{*

*if (i%j===0)*

*{*

*count++*

*}}*

*if (count===2)*

*{*

*clg(i)*

*}*

*}*

# Functions in JavaScript

* *a block of code which is used to perform the specific task*
* *By using functions, we can achieve* ***code re-usability*** *and* ***code modularity***
* ***Code re-usability*** *> Reusing the same code again and again without writing it*
* ***Code Modularity*** *>> Dividing the bigger task into small modules*
* *In JavaScript, function can be declared with the help of function keyword.*
* *We can also declare the function without the function keyword.*
* *In JavaScript functions, there* ***is no strict return type.*** *i.e, we can return any kind of data. If we don’t*

*return anything from the function, by* ***default it returns Undefined***

* *We can pass the parameters at the time of declaration, but we should not use the keywords to declare the parameters*
* *the* ***function will not be executed as long as it’s not called***

##### Syntax with an example:

***Syntax >> function functionname (Parameters)***

##### {

***}functioncall();***

##### Example >>

function add(a,b) // we can passs values of a and b

{

console.log(a+b);

}

add(a,b);

function add(a,b=1) // b is default parameter, a is not a default parameter

{

console.log(a+b);

}

add(a,b);

##### Note >>

* *Better practice is to assign the default values to the parameters at the time of declaration, whenever we call the function with no values it takes undefined, it over comes the problem of it by taking the default values that we assigned.*
* *If we pass the values to the parameters, it will override the default values.*
* *The parameters on the function are considered as the local variable of the functions*
* *At the time of declaration if we pass variable to the function, we call it as Parameters of the function.*
* *At the time of calling the function, if we pass values to the parameters, we call it as Arguments*

# Types of functions in JavaScript

##### Named function:

* *A function which is* ***declared with a name is known*** *as named function.*
* *Function name holds function structure or reference of the function*
* *If we are printing the function name. we get the structure of the function as it is stored in the variable*
* *Whenever we call the function and assign to the variable, returned value will be stored inside the variable*
* *Whenever we call the function and print, it prints the return value*
* *Whenever we print the function name, it prints the structure*
* *If return statement is not present, it returns undefined*

##### Example :

***Example Point 2>>***

function m1()

{

let a = 20; console.log(a)

}

function m2()

{

let a = 20; console.log(a)

}

***Ex point 1 >>***

function add(a,b)

{

let sum = a+b; console.log(sum)

}

add();

***Note:*** *let res =m2 // m2 and res structure will be same (if we check it prints True) but m1 !== m2 as the var is diff and it creates a different object*

###### Important:

* *Return statement must be the last statement of the function. If we pass any other statement*

*after return statement then the statement will be unreachable. It won’t get executed*

* *Function will be popped out from the stack when there is no statement to execute and when it sees return statement*

##### Function Expression

* *Any* ***function structure assigned to a variable*** *is known as function expression.*
* *It can be Anonymous, named or arrow function. Same as before example.*

##### Example Point 2>>

***let x=function m1() // Function expression***

*{*

*let a = 20; console.log(a)*

*}*

##### 3.Anoynumous function

* *A function which is declared without a name is known as anonymous function*
* *This function must be* ***assigned to a variable*** *or must be* ***passed as an argument*** *to another function*

##### Note :

* *Whenever we pass a function as an argument to another function, then we should use anonymous function or arrow function*

##### Ex for point 1>>

***Ex for point 2>>***

*function add = (a,b)*

*{*

*clg(a); clg(b)*

*}*

*add();*

*a = ()*

*{*

*Clg(“passing a function”)*

*}*

let a = function()

{

clg("Hello");

} a();

##### Arrow function (Lambda or Fat arrow)

* *It is used to reduce the code*
* *it is a syntactic sugar for writing functions*
* *When we write arrow function, function keyword should not be used*
* *If the function is having one parameter, then the parenthesis is not mandatory. i.e parenthesis are optional.*
* *When the function is having no parameters or more than one parameter, then parenthesis is mandatory*
* *When a function block is having more than one statement, then flower braces is mandatory as well as return statement is mandatory*
* *When a function block is having only one statement, then curly braces is optional and return keyword should not be used.*
* *It avoids "****this****" keyword conflict*
* *They are very useful when the function is having only one statement*
* *In arrow function if the one statement is the return statement then keyword should not be used.*

##### NOTE>>

*Whenever we need the* ***return value of function****, either* ***call the function and print or store it in a variable. Examples::***

##### 1. passing one parameter

***2. Passing two parameters.***

let add = function(a,b)

{

return a+b;

}

>> conversion

let add = (a,b)=>a+b clg(add(a+b))

let add = function(a)

{

return a;

}

>> conversion

let add = (a)=>a clg(add(a))

##### 3. Passing multiple statements.

let add = function(a,b)

{

clg(a)

clg(b) return a+b;

}

>> conversion let add = (a,b)=>

{

clg(a)

clg(b) return a+b;

}

clg(add(a+b))

##### Callback Function

* + *It is function which is passed as an argument to another function*
  + *It is used to handle* ***asynchronous type of data*** *(Will be taught later)*

***Example*** *>> add* ***(function()***

***{***

Passing the function as argument

***return 22;***

***}****,1)*

=>> add(23,()=>34)

##### Higher Order Function

*It is a function which* ***accepts function as an argument*** *or* ***returns a function.***

***Example 1>>***

Example 2 >>

function sub(a,b)

{

return function()

{

clg(a-b)

}

}

function add(func, b)

{

clg(func) clg(b)

}

add(()=>3) //

clg(sub) >> prints the structure

clg(sub(function ())) >> gives the return value or we can store it in a variable and then we can print it back

let res = sub(3,4) clg(res);

##### Self-invoking function

* + *IIFE stands for immediately invoked function expression and also known as self-invoking function*
  + *IIFE function will be called immediately*
  + *IIFE function is used to protect the data*
  + *IIFE function will be called only once*
  + *No return statement is used*

***Example 1 >>***

***Example 2>***

( (a,b)

{

clg(a+b)

})(2,3);

( function add(a,b)

Function enclosed in ()

{

clg(a+b)

})add(2,3);

##### Currying

* + *Whenever the function is returning another function then we can chain the functions and invoke*
  + *It’s also known as functional chain*
  + *Currying is possible in higher order function*

***Example >>***

function a(p)

{

return function (q)

{

return function (r)

{

clg(p+q+r)

}

}

}

***O/P>>***

a(10)(9)(6) // Prints the output as 25 a(10)(9)(6)()

# Strings in JavaScript

* *It is a collection of character which is enclosed in " " , ' ' , ` `*
* *The string which is enclosed in " " and ' ' is known as string literals*
* *The string which is enclosed in ` ` is known as template literals or template strings*

***Difference Between String Literals and Template Strings***

|  |  |
| --- | --- |
| ***String Literals (“ “ ‘ ‘ )*** | ***Template Strings ( ` ` )*** |
| * *multi line string is not possible* | * *We can easily write multi line strings* |
| * *Identifiers cannot be replaced* | * *Identifiers can be repalced inside ` ` by using place holder>> ${ JSCODE}* |
| * *Place holder cannot be used* | * *Place holder can be used* |
| * *It is introduced in JavaScript* | * *It is introduced in ECmascript 6* |

***Note : Place holder is replacing the value Example 1 (Multiline strings)*** *>>*

*let s = "welcome to*

*Jaspiders "*

*clg(s) // throws an error incase we are using " " >> Doesnot print in multiple lines*

*let s = `welcome to*

*Jaspiders `*

*clg(s) // it prints incase we are using ` ` >> print in multiple lines*

##### Example 2 (Place holders)>>

*console.log(boy+ "and " +girl "are going to "+place + "by" +vehicle) >> It can be replaced by placeholder*

*console.log****(${boy}*** *and* ***${girl}*** *are going to* ***${place}*** *by* ***${vehicle}***

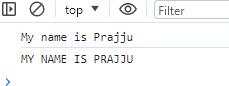
# String Methods

###### toUpperCase()

* + *used to convert the string to uppercase and doesn't have any argument.*

###### Example >>

*let name = "My name is Prajju" console.log(name);*



*let uppercase =* ***name.toUpperCase();***

*console.log(uppercase)*

* ***toLowerCase()***
  + *used to convert the string to lowercase and doesnot have any argument****s.***

***Example >>***

let name1 = "MY NAME IS PRAJJU"

console.log(**name1.toLowerCase())**



* ***trimStart( )***
  + *Used to* ***remove the spaces at the starting of the string*** *and doesn’t have any*

argument.

***Example*** *>>*

// trimStart()

let name = "

My name is Prajju"

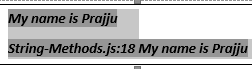
console.log(**name.trimStart())**



##### trimEnd()

* + *Used to remove the spaces at the ending of the string and doesn’t have any argument.*

***Example >>***



let name = "My name is Prajju

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

"

* ***trim()***
  + *Used to remove the spaces at the beginning and at the end of the string and doesn’t*

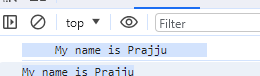
have any argument.

let name = "

My name is Prajju "

console.log(name);

console.log(**name.trim())**



##### concat(arg1,arg2..)

* + *Used to join 2 or more strings and accepts n no. of args where the args are strings to be joined.*

***Example >>***

let a = "My name is Praj and" let b = " I live in Bengaluru" **console.log(a+b)**



##### replace(arg1,arg2)

* + *used to replace old string with new string and accepts 2 args where the 1st arg is old and 2nd is new string (1st will be replaced by 2nd****). Replaces only 1st occurrence.*** *If you need to replace all occurrence, then we* ***should use 1st arg as regular expression.***
* ***Example:***



***1st value to be replaced (Old value ) value that will be replaced only 1st occurrence***

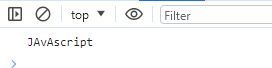
let subject = "Javascript" console.log(subject.replace("a", "A"))

##### replaceall()

* + *used to replace old string with new string at all the occurrence.*

***Example >>***

let subject = "Javascript" console.log(subject.replaceAll("a","A"))



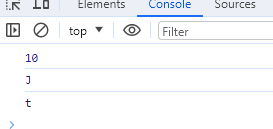
##### String length

* + *Whenever we declare a string every character of a string will be stored in indexes.*
  + *The 1st char will be stored in 0th index and the last char will be stored in length-1 index.*
  + *The length of the string is equal to the total no. of characters in the string*
  + *The length of the string can be calculated by using length property.*
  + *Whenever we perform any operations(inbuilt) on the string, it always works on forward direction*
  + *If we want to perform operations from the end, we use -ve indexes which starts from -1.*

***Example >>***

let word = "Javascript" console.log(word.length) console.log(word[0])

console.log(word[9])



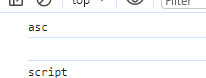
##### slice(arg1,arg2)

* + *Slice method is used to cut the part of the string.*
  + *It accepts 2 args where the 1st arg is start index and the 2nd arg is end index*
  + *1st arg is included and the last arg is excluded*
  + *If we pass only one arg, it will consider it as a start index and cuts the whole string till the end from the given index*
  + *It returns part of the string.*
  + *It always works on the right direction. If the indexes are not matching, then it returns empty string*

***Example:***

let word = "Javascript" console.log(word.slice(3,6)) console.log(word.slice(6,2))

console.log(word.slice(4))



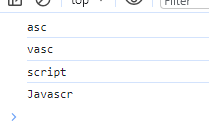
##### substring(arg1,arg2)

* + *It is same as slice method but considers -ve indexes as 0.*

##### Example :

let word = "Javascript" console.log(word.**substring**(3,6)) console.log(word.**substring**(6,2)) console.log(word.**substring(4))**

**console.log(word.substring**(-4,7))



* ***padstart(arg1,arg2)***
  + *This method is used to join a new string at the start of the string*
  + *The 1st arg takes the number value and the 2nd value takes the string*
  + *When the function is called, it compares the 1st arg (number value) with the string length. If the number value is greater than the string, then it adds the string that is present in arg 2 at the beginning till it covers the actual string.*
    - ***Scenario 1>> If its +ve number***

*Then its going to join at the start of the string.*

* + - ***Scenario 2>> If its -ve number or 0***

It returns the actual string

* + - ***Scenario 3>> If its +ve and String size is more than the length value,***

it just occupies for the remaining value

##### Example :

let word ="Javascript" console.log(word.padStart(15,"abc"))

console.log(word.padStart(12,"Hello"))



* ***padEnd(arg1,arg2)***
  + *This method is used to join a new string at the end of the string*
  + *The 1st arg takes the number value and the 2nd value takes the string*
  + *When the function is called, it compares the 1st arg (number value) with the string length. If the number value is greater than the string, then it adds the string that is present in arg 2 at the end till it covers the actual string.*
    - ***Scenario 1>> If its +ve number***

*Then its going to join at the end of the string.*

* + - ***Scenario 2>> If its -ve number or 0***

It returns the actual string

* + - ***Scenario 3>> If its +ve and String size is more than the length value,***

it just occupies for the remaining value

##### Example :

let word ="Javascript"

console.log(word.padEnd(15,"abc")) console.log(word.padEnd(12,"Hello"))



##### charat(arg1)

* + *This method is used to get the character based on index position*
  + *It accepts one arg of type number where the number is index position*
  + *It returns a char on available index. If index position is not available, then it returns empty string*

***Example :***



let word = "JavaScript"

console.log(word.charAt(6)) console.log(word.charAt(17))

* ***chartcodeat(arg1)***
  + *This method is used to get the ascii value of character based on index position*
  + *It accepts one arg of type number where the number is index position*
  + *It returns a ascii value of char on available index. If index position is not available, then it returns NAN*

***Example :***



let word = "JavaScript". console.log(word.charCodeAt(6))

console.log(word.charCodeAt(11))

***Note : Method used to get the ascii value of character "i".charcodeat() // 105 >> in place of i we can pass any character***

* ***stringfromcharcode(arg1)***
  + *This method is used to get the character based on ascii value*
  + *It accepts one arg where the arg is a value*
  + *It returns the char of the given return value*

***Example :***

* ***at(arg1)***
  + *It is same as charat method but it accepts -ve values.*
  + *It returns the char available on given index for both +ve and -ve indexes.*

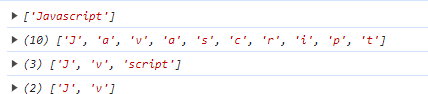
***Example :***

let word = "Javascript" console.log(word.at(2))

console.log(word.at(-2))

* ***split(arg1,arg2)***
  + *It is used to split the string based on n given string*
  + *It accepts 2 args where the 1st arg is the string based on which we are splitting and the 2nd arg is the limit of split*
  + *It returns array of splatted strings where the splatted strings will be considered as the elements of array*
  + *If we don’t pass the limit, it splits the string for all the occurrence of a given string*
  + *If we pass the limit, it splits based on the limit. for ex; if the limit is 2, it performs only 2 strings*
  + *If we want to split every char of the string then, we pass empty string as 1st arg*
  + *If we don’t pass any arg, the whole string will be splatted*

***Note :: This method converts from string to an array Example :***



let word = "Javascript" console.log(word.split()) console.log(word.split("")) console.log(word.split("a"))

console.log(word.split("a",2))

* ***startswith(arg1)***
  + *startswith() is used to check whether the given string is starting with specific string or not*
  + *If the string is stating with a specific string, then it returns TRUE else returns FALSE*

***Example :***



let word = "Javascript" console.log(word.startsWith("j"))

console.log(word.startsWith(""))

***Example :***

* ***Endswith()***
  + *endswith() is used to check whether the given string is ending with specific string or not*
  + *If the string is ending with a specific string, then it returns TRUE else returns FALSE*

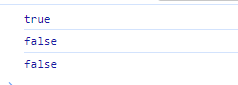
let word = "Javascript"

console.log(word.endsWith("pt")) console.log(word.endsWith(""))



* ***includes(arg1,arg2)***
  + *this method is used to check whether the given string is available or not inside actual string*
  + *This method accepts 2 arguments where the 1st arg is string to be searched and the 2nd arg is index position from where to search*
  + *If we dont pass the 2nd arg its going to search the string from the 0th index*
    - *If we pass the 2nd arg it is going to search the string from the particular index*
    - *If the string is available, it returns true else it returns false*

***Example :***

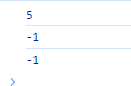


let word = "Javastring" console.log(word.includes("a")) console.log(word.includes("z"))

console.log(word.includes("a",5))

* ***index(arg1,arg2)***
  + *This method is used to check whether the given string is available or not inside actual string*
  + *This method accepts 2 arguments where the 1st arg is string to be searched and the 2nd arg is index position from where to search*
  + *If we dont pass the 2nd arg its going to search the string from the 0th index*
  + *If we pass the 2nd arg it is going to search the string from the particular index*
  + *It returns the index position of the string if its available else it returns -1. It always returns 1st occurred string index position*

***Example:***



let word = "Javastring" console.log(word.indexOf("a")) console.log(word.indexOf("z"))

console.log(word.indexOf("a",5))

* ***lastindex(arg1, arg2)***
  + *It is same as index( ) but it works in reverse order.*
* ***Search()***
  + *This method is same as indexof( ) but it doesn't have 2nd arg*
  + *It searches the string based on regular expression*
  + *the 1st arg can be a regular expression or string*

***example :***



let word = "Javascript"

console.log(word.search("p"))

* ***match(arg1)***
* ***matchall(arg1) to b taught later***

# JavaScript Arrays

* *They are used to store the values of same type and multiple type.*
* *In JavaScript, the array values will be stored in index position*
* *The values of the array are also known as elements of the array*
* *The 1st element will be stored in 0th index position and the last element will be stored in length- 1 position*
* *Whenever we try to* ***access the array element which is not stored in some index, then we get undefined***
* *In JS array, size is not fixed*
* *In JS array, we can store both primitive and non-primitive type of data*
* *In jS array, manipulation of array elements is very easy*
* *It is having lot of built in methods*

***Note :***

* + *In JS array we can skip the indexes but its not a good practice*
  + ***Do not use delete operator to delete the elements as it leaves the loop holes at the indexes where delete is performed and maintains same length of the array***

***Create or update values***

**a[2] = "Dummy" //** replaces a[2] by dummy. if the position is not present, it creates a new element.

##### Functions on Arrays

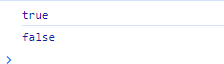
* ***includes(agr1,arg2)***
  + *Searches for the element, if its present, returns true else returns false*

***Example :***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.includes("watch")) console.log(a.includes("mobile", 6))

***Output***



* ***indexof(arg1,arg2)***
  + *returns the index value if present, else returns -1*
  + *works from left to right direction*
  + *1st arg is the element that we need and the 2nd arg is from where to begin the search*

***Example:***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"] console.log(a.indexOf("mobile"))

console.log(a.indexOf("mobile", 2))

***Output :***



* ***lastindexof(arg 1, arg 2)***
  + *Works similar to the index(arg1, arg2)*
  + *returns the index value if present or else returns -1. Works from right to left*

***Example :***

a.lastindexof(watch,2) // 1

a.lastindexof(watch) // 4

a.lastindexof(earphone, 4) // 2

a.lastindexof(iphone) // -1

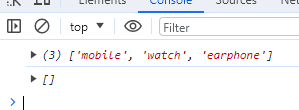
* ***slice(arg1,arg2)***
  + *it returns the part of the string where the 1st arg is included and the last arg is excluded where the arg values should be the index values of the array.*

***Example :***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"] console.log(a.slice(0,3))

console.log(a.slice(4,4))

***Output//***



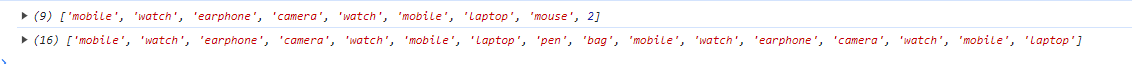
* ***concat(arg1,arg2...)***
  + *It adds the elements to the array.*

***Example :***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"] console.log(a.concat("mouse",2))

console.log(a.concat(["pen","bag"],a))

***Output***



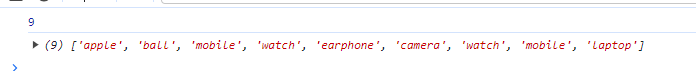
* ***Unshift(arg1 /.)***
  + *It is used to add elements to the array*
  + *It always adds the elements at the starting of an array*
  + *It accepts n no. of args where the args are new elements to be added.*
  + *This method returns the updated array length. It affects* ***original array***

***Example :***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.unshift("apple", "ball")) console.log(a)

***Output:***



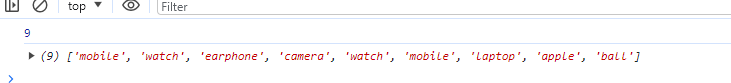
***Example:***

* ***push(arg1 )***
  + *It is used to add the elements to the array*
  + *It always adds the elements at the end*
  + *It accepts n no. of args where the args are new elements to be added*
  + *This method returns the updated array and affects the original array*

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.push("apple", "ball")) console.log(a)

***Output:***



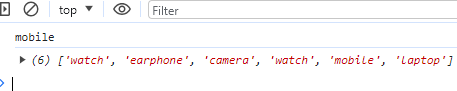
* + - ***shift()***
      * *It is used to remove the 1st element from the array*
      * *It doesn't have any args. It returns the removed element*

***Example :***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.shift()) console.log(a)

***Output :***



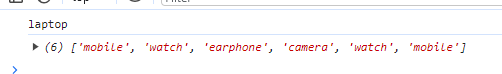
* ***pop()***
  + *It is used to remove the last element from the array*
  + *It doesn't have any argument*
  + *It always removes the last element*
  + *It returns the removed element from the array*
  + *It affects original array*

***Example :***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"] console.log(a.pop())

console.log(a)

***Output :***



* ***Splice(arg1, arg2 )***
  + *It is used to add or delete the elements from the array based on the index position*
  + *It accepts n no. of args where the* ***1st arg is the index position*** *from where to add or delete and the* ***2nd argument is no, of elements to be deleted From the 3rd arg all the args are considered as new elements to be added***
  + *If we dont want to add the elements, pass only 2 args*
  + *If we dont want to delete the elements, then pass 0 as zero arg*
  + *It* ***returns an array of removed elements***
  + *It affects original array*
  + *If we dont delete any element, it returns empty array*
  + *If we pass only one arg, it deletes all the elements from the given index*

***Examples***

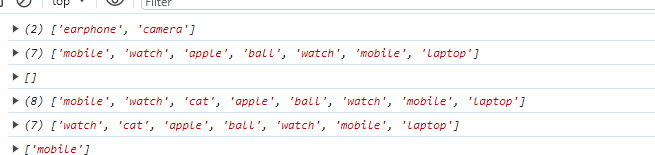
let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.splice(2,2,"apple", "ball")) console.log(a) console.log(a.splice(2,0,"cat")) console.log(a)

console.log(a.splice(1))

console.log(a)

***Output***



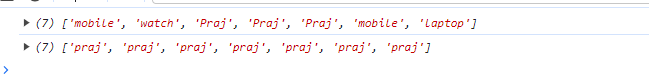
* ***fill(arg 1, arg2...)***
  + *it is used to fill all the array elements with a new value*
  + *It accepts 3 args, where the 1st arg is value to be filled and the second arg is the start index position from where to fill and the 3rd arg is the end till where to fill*
  + *Start index in included and the end index is excluded*
  + *If we dont pass the 3rd arg, it fills till the end of the array from the given index*
  + *If we pass only one arg it fills the whole array*
  + *It returns an array with newly filled elements*
  + *It affects original array*

***Example :***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.fill("Praj", 2, 5)) console.log(a.fill("praj"))

***Output***



* ***Reverse()***
  + *Prints the elements of the array in the reverse order*

Example:

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.reverse())



* ***join("arg 1")***
  + *It is used to separate the elements with the particular separator*
  + *It accepts one argument of type string*
  + *If we dont pass any argument, it separates array elements.*

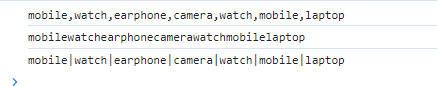
***Note : It converts array to string Example :***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.join()) console.log(a.join(""))

console.log(a.join("|"))

***Output***



* ***at(arg1)***
  + *It takes 1 arg. i.e, the index value of the array and prints the element that is present in that corresponding index.*
  + *It returns undefined when there is no element present at that index*

***Example:***

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(a.at(1))

***Output***



* ***foreach(arg1)***
  + *It’s actually iterating for an array of elements.*
  + *It accepts one arg as a callback function.*
  + *We can pass max of 3 parameters inside the function.*
  + *The 1st arg is the element, 2nd is the index number and the 3rd prints the whole array.*
  + *The return value is always "****Undefined****"*

***Example:***

let details = [ "Jam Josh" ,"Justina Kap" ,"Chris Colt","Jane Doe"];

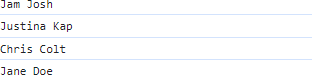
details.forEach((names)=>

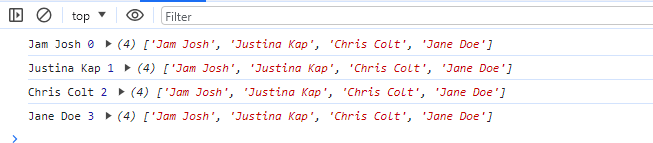
{

console.log(names)

})

***Output***





let details = [ "Jam Josh" ,"Justina Kap" ,"Chris Colt","Jane Doe"];

details.forEach((names, ind, array)=>

{

console.log(names, ind , array)

}, )

* ***Map()***
  + *Its actually an iterating for an array to make the* ***changes of an element in array****.*
  + *It accepts one arg as a callback function.*
  + *We can pass max of 3 parameters inside the function. The 1st arg is the element, 2nd is the index number and the 3rd prints the whole array.*
  + *Map() returns an array of updated elements*

***Example :***

let details = [ "Jam Josh" ,"Justina Kap" ,"Chris Colt","Jane Doe"];

console.log( details.map((name)=>

{

return name.toUpperCase()

}

))

***Output:***



* ***Filter()***
  + *It’s actually an iterating for an array of elements*
  + *. It accepts one arg as a callback function.*
  + *We can pass max of 3 parameters inside the function. The 1st arg is the element, 2nd is the index number and the 3rd prints the whole array.*
  + *This method returns an array of filtered elements.*
  + *This method stores the elements inside the new array if the condition is satisfied else it*

doesn’t store the elements

* + *If none of the elements satisfies the condition, it returns empty array*

***Examples***

let num = [1,5,4,7,7,9,3,5,2]

console.log(num.filter((number)=>

{

return number>4;

}

))

***Output:***

* ***Some()***
  + *It is used to check at least one element is satisfying the condition or not.*
  + *It accepts one arg as a callback function.*
  + *It returns True if any one element is satisfying the condition else it returns false.*
  + *Once the condition is satisfied, its going to terminate the loop****.***

Example:

let num = [1,5,4,7,7,9,3,5,2]

console.log(num.some((number)=>

{

return number == 7;

}))

***Output***

***// true***

* ***Every()***
  + *It accepts one arg as a callback function.*
  + *It is used to check whether every element is satisfying or not.*
  + *It returns true. If all the elements are satisfying the condition, then TRUE else returns false.*
  + *It checks the condition for all the elements in the array Example :*

let num = [1,5,4,7,7,9,3,5,2]

console.log(num.every((number)=>

{

return number == 7;

}

))

***Output***

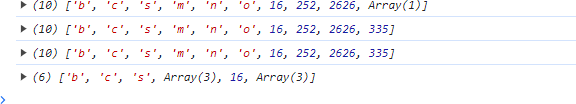
* ***Flat()***
  + *It is used to flatten the array (It joins nested array elements to the existing array)*
  + *Flat method accepts one arg of type number*
  + *the arg is depth of flat. i.e, how much depth we should flatten the array*
  + *If we dont pass any arg Its going to flat one depth*
  + *It returns the flattened array*
  + *It doesn’t affect the original array*

***Example***

let a = ["b","c", "s", ["m","n","o"], 16, [252, 2626, [335]]]

console.log(a.flat(1)) console.log(a.flat(2)) console.log(a.flat(3)) console.log(a)

***Output***



* ***Flatmap()***
  + *It’s a combination of flat and map method with level 1*
  + *It is used to flatten the array (It joins nested array elements to the existing array)*
  + *Flat method accepts one arg of type number*
  + *the arg is depth of flat. i.e, how much depth we should flatten the array*
  + *If we don’t pass any arg Its going to flat one depth*
  + *It returns the flattened array.*

Note >> update + array elements

***Example >>***

let a = ["b","c", "s", ["m","n","o"],"s"] let res= a.flatMap((ele)=>

{

return ele

}

)

console.log(res)

console.log(res.map(((abc)=>

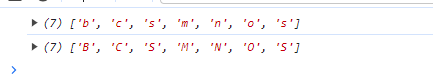
{

return abc.toUpperCase()

}

)))

***Output***



* ***Find(arg1)***
  + *It is used to get the element which is satisfying the condition*
  + *It accepts one arg as the callback function.*
  + *When the condition satisfies, it returns element else returns undefined.*
  + *When condition satisfies, its gonna terminate the loop****.***

***Example:***

let num = [1,5,4,7,7,9,3,5,2]

console.log(num.find((ele)=>

{

return ele == 7

}

))

***Output***



* ***Findindex()***
  + *It is used to get the index position of the element which is satisfying the condition*
  + *It accepts one arg as the callback function.*
  + *When the condition satisfy, it returns index of an element else returns -1.*
  + *When condition satisfies, its gonna terminate the loop*

***Example***

let num = [1,5,4,7,7,9,3,5,2]

console.log(num.findIndex((ele)=>

{

return ele == 7

}

))

***Output***



* ***Sort***
  + *It is used top sort the elements of an array based on Ascii values by default.*
  + *It accepts 1 arg where the args are callback function*
  + *If we dont want to sort based on ascii values we should pass the arg to the function*
  + *Callback function accepts 2 args, where the args are array elements*
  + *If we want to sort in ascending order, we should return (element 1)- (element 2) and for descending its vice versa*
  + *If it’s a number pass the callback function with 2 parameters (a-b) for ascending and b-a for descending. If its for string just sort() will work*

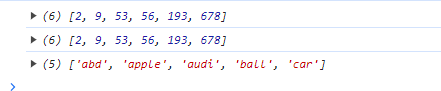
***Example:***

let num =[2,56, 678, 9, 53, 193]

console.log(num.sort((a,b)=>a-b)) console.log(num)

let a = ["apple","car", "ball", "abd", "audi"] console.log(a.sort())

***Output:***

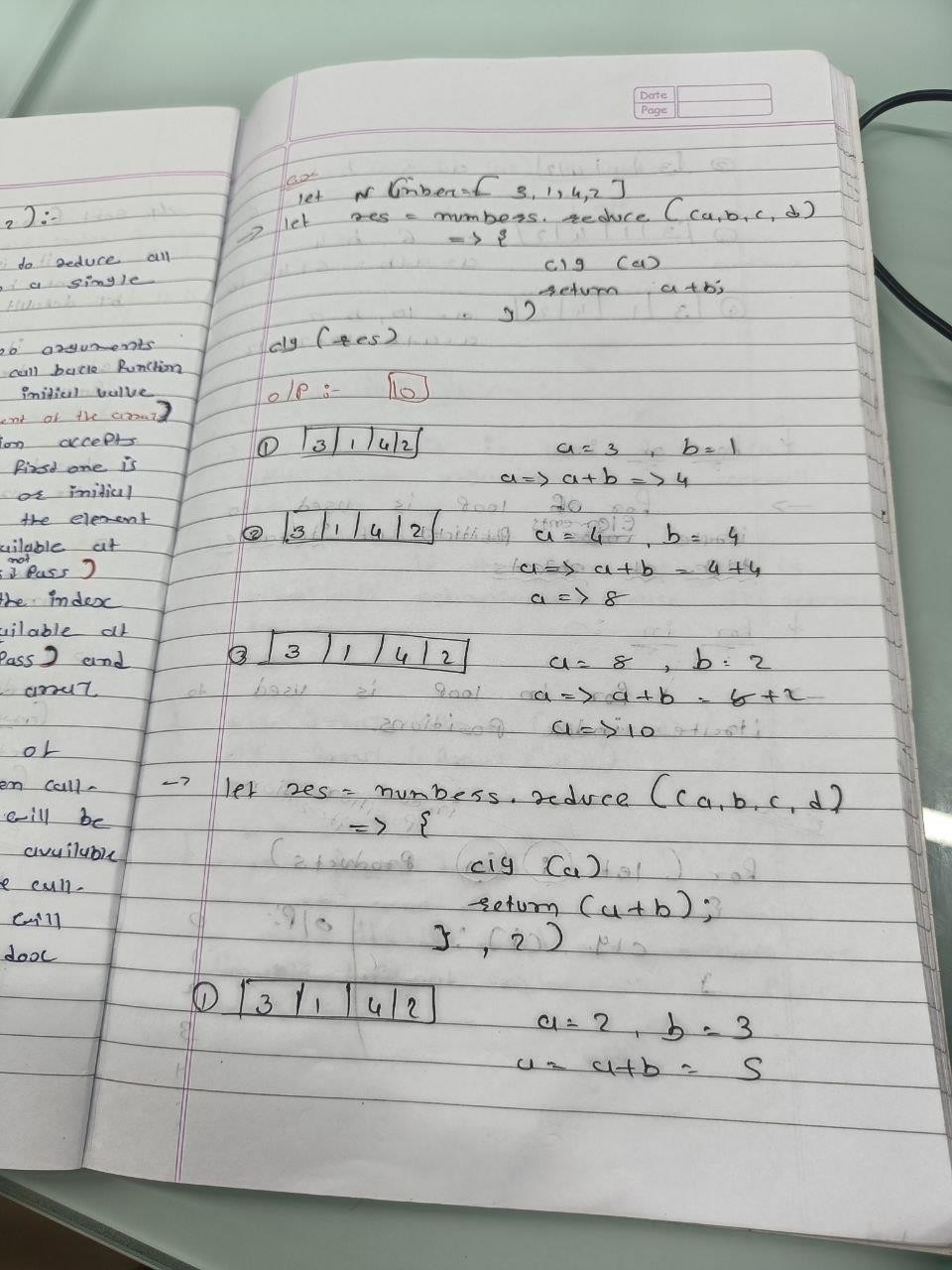


***Note: Sort works for string >> Either should be upper case or lower case. If its not, use map function and convert to the either one.***

***Reduce(arg1, arg 2)***

* ***Arg1 should be a callback function***
* ***If arg2 is passed, it considers as the first value. If 2 arg is not passed, consider the below example.***

***Let n = [3,1,4,2]***



* ***For of:***
  + *It is used to iterate the elements of iterable type of objects*

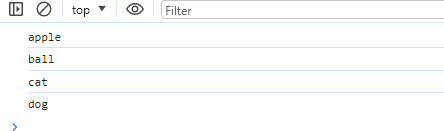
***Example :***

let a = ["apple", "ball", "cat", "dog"] for (const ele of a) {

console.log(ele)

}

***Output:***



* ***For in***
  + *It is used to iterate the Index of the elements*

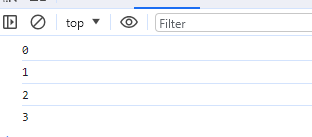
Example :

let b = ["apple", "ball", "cat", "dog"] for (const ele in b) {

console.log(ele)

}

Output



* ***Array.from()***
  + *It is used to convert the iteratble type of object to a pure array*
  + *It accepts one argument of type Iterable object*

***Example:***

let a = "Javascript"

console.log(Array.from(a))

***Output :***



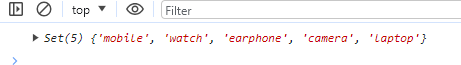
* ***Set()***
  + *It is an object in Javascript which is used to delete a duplicate element in array*
  + *It accepts one arg where the arg is iterable type of object*

***Example*** *>*

let a = ["mobile","watch" , "earphone", "camera", "watch", "mobile", "laptop"]

console.log(new Set (a))

Output



# Objects

* + *t is a real world entity which has physical existence*
  + *It is represented by { }*
  + *In JS, object is a collection of key and Value pairs*
  + *In JS object key's should not be repeated, if the key is repeated last occurred key value will be considered*
  + *If we want to access the values of the object, we use object name. keyname or object["keyword"]*
  + *In JS object, key and value must be separated by :*
  + *In JS object, key-value pair must be separated by ,*
  + *In JS object value can be any kind of data*

##### Note >>

* *It donot have index position and hence it is not iterable*
* *If we dont have Keys in the object, and if we try to access the key we get undefined*

***Syntax with an example.***

***Let student = {name : “ praj “, age : 24}***

***// CRUD operations on Objects***

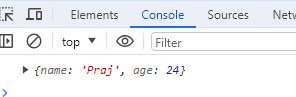
* ***Update or create.***
  + *It takes the same syntax as to create and update.*
  + *If the element is not present it with the same key name, it creates a new pair of elements, if the key element is present, it updates the value*

***Example :***

let student = {name: "Praj", age :24}

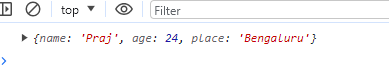
console.log(student)

***Output ::***



* ***Creating place***
* let student = { name: "Praj", age :24}
* student.place="Bengaluru";
* console.log(student)

***Output***



* ***Read()***
* *We can access the value with the help of keyname.*

***Example:***

let student = { name: "Praj", age :24, place:"Bengaluru"}

console.log(student.place)

***Output***



* ***Delete()***
  + *We can delete the object with the help of keyname.*

***Example:***

let student = { name: "Praj", age :24, place:"Bengaluru"} delete student.age

console.log(student)

***Output :***



***Passing a function inside an object***

let student = {name : "praj", age:25, dept:"ise", gender:"male", getdetails:function()

{

// console.log(this)

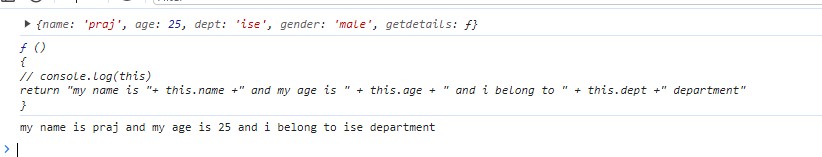
return "my name is "+ this.name +" and my age is " + this.age + " and i belong to " + this.dept +" department"

}

}

console.log(student) console.log(student.getdetails)

console.log(student.getdetails())



* ***Nested Object***
  + *Object inside an object*

***Example :***

let student = {name:"praj", age:25, dept:"ise", gender:"male",

hobbies:{play:"cricket",watch:"web-series"}} console.log(student.hobbies.play)

***Output***



***Note >>***

1. ***When we try to access undefined key, it throws undefind.***
2. student = { name: "Praj", age :25, gender:"male"}
3. console.log(student.marks)

***Output Undefined***

1. ***When we are trying to access the undefined key's property, it throws an error. In order to avoid that use ? at the key.. >> It executes like if the key is present, it executes that or else prints undefined. The same tis goes for null if its defined in the marks***
2. student = { name: "Praj", age :25, gender:"male"}
3. console.log(student.marks)
4. console.log(student.marks.math)

***Output***

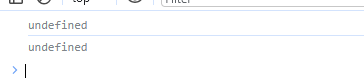


***How to overcome***

student = { name: "Praj", age :25, gender:"male"} console.log(student.marks)

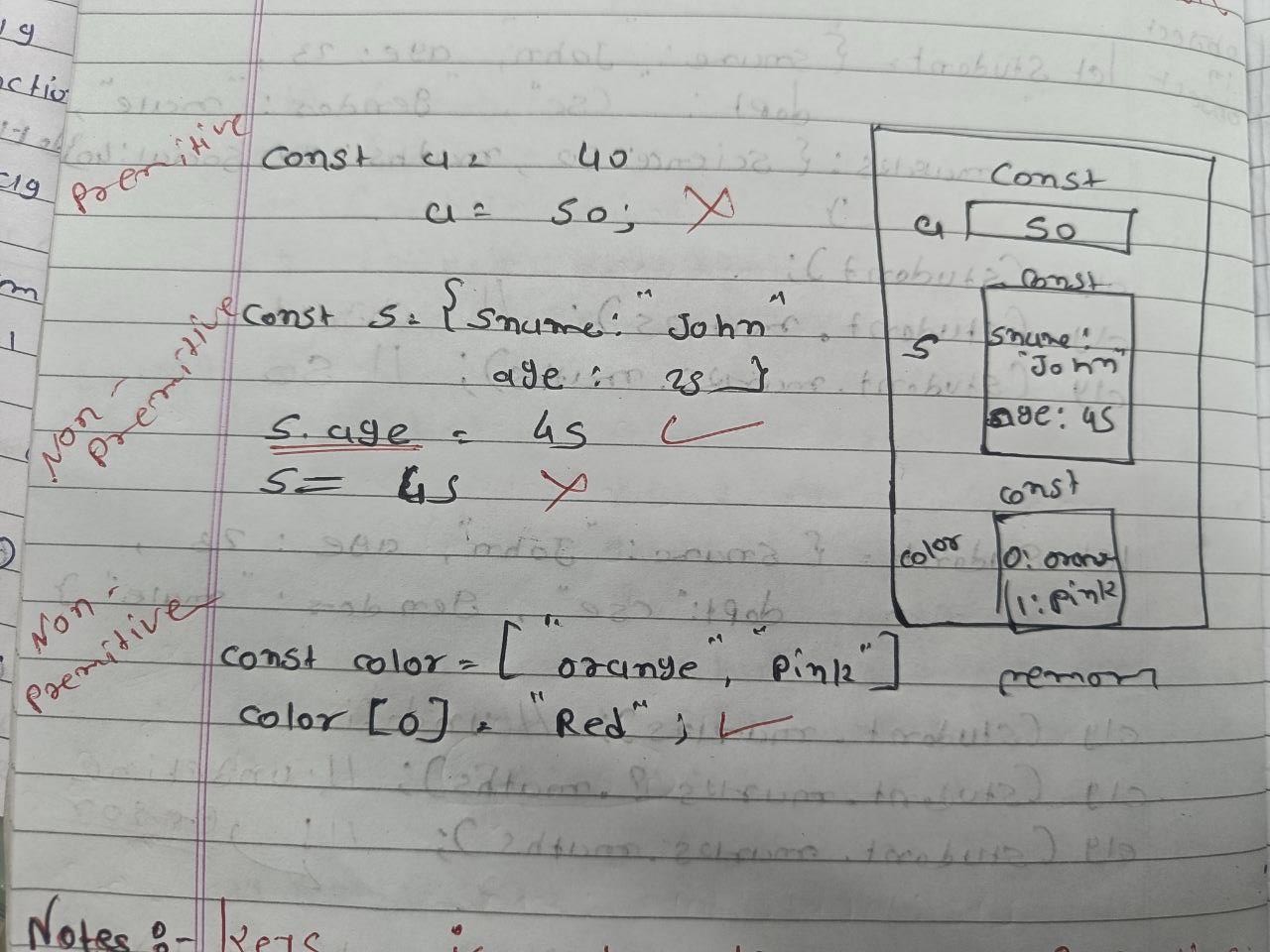
console.log(student.marks?.math)

***“?”***



***Note :***

* ***If we are using const keyword for non-primitive datatypes, it affects if the variable names are same. If the update is made on the objects or on the elements, it will work. (EXAMPLE 1)***
* *For -in loop works as it prints the key values but for of loop does not work as it works Only on the iterable type of objects. (example 2)*



***Example 2 :***

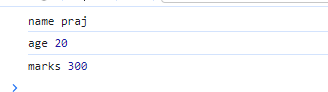
let student = {name:"praj", age:20, marks:300} for (const i in student)

{

console.log(i,student[i])

}

***Output***



***Shallow copy and deep copy***

* ***Deep Copy***
  + *In deep copy, it works on the primitive data types. Where it only copies the value from one variable to another and disconnects the reference. Deep copy stores copies of the object's value*

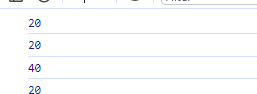
Example :

let a =20; let b=a;

console.log(a) console.log(b)

a = 40;

console.log(a) console.log(b)

***Output***

***After updating the value of a, its affecting only a and not b as it disconnects the difference.***

* ***Shallow copy***
  + *In Shallow copy, it works on the non-primitive data types. Where it only copies the value and points to the same address.* Shallow Copy stores the references of objects to the original memory address

***Example :***

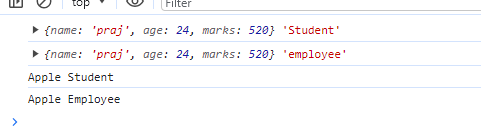
let student = {name:"praj", age:24, marks:520} let emp = student;

console.log(student, "Student") console.log(emp,"employee")

student.name = "Apple"

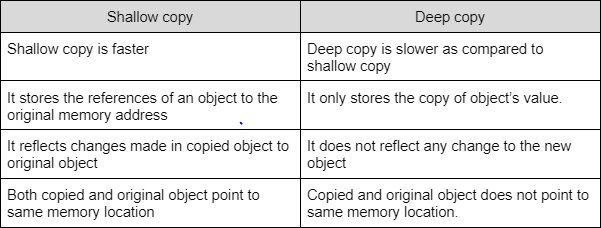
console.log(student.name, "Student") console.log(emp.name, "Employee")

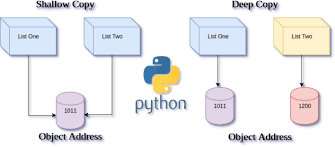
***Output:***



*Whenever we are updating the value in the student, its even updating in the employee as its pointing to the*

*same reference address. It’s a major disadvantage, in-order to overcome this, we use spread operator.*





***Converting shallow copy to deep copy***

for Objects

let student = {name:"praj", age:24, marks:520} let emp = {...student};

s.age=45;

clg(student )// updates the age >> 454

clg(emp) // stays 31 only

let colors = ["red", "orange"] let favcolors = [...colors]

favcolors.push["black"]

clg(colors) // doesnot add the colors clg(favcolors) // does add colors

***Note >>***

***For arrays >> the deep copy can be achieved by the following methods***

***let favcolors=[...colors]***

***let favcolors = colors.slice(0) let favcolors = colors.concat()***

***let favcolors=colors.map(ele=>ele) let favcolors = colors.filter(ele=>true)***

* ***Spread Operator***
  + *It actually creates a new array or object and it passes the elements or the objects from the old nan -primitive data types to the new array or objects as the separate entity. Above is the example.*

It works only for the 1st level of the object . If there are nested object, it effects both the objects(Example is below shown)

let student = {name : "praj", age : 24,

marks: {maths : 99, science:92}, address:{city : "Banalore", state: "karnataka",

mobile:{personal : 123456789, official : 9887654321}}}

let employee = {...studentt}

clg("STUDENT", student) // Prints the entire object of student clg("EMployee", employee) // Prints the entire object of the student

employee.age = 45; // update sonly the employee age

employee.marks.maths = 300 // updating both the values at the student and the employee

to convert the data format>>

let jsonstudent = JSON.stringify(student)

* ***JSON>> Javascript Object Notation***
  + *IT is used in creation in of the API's*
  + *API(Application Programming iterface) >> Takes request from client and gives response to client by checking with the server.*
  + *API's will be in the form of JASON data*
* ***JSON***
  + *Key should always be enclosed in " "*
  + *It should be stored in .json file*

*{*

*"type" : phone*

*" price": 400*

*"features" : ["Camera", "Wifi"]*

*}*

* ***JS Object***
  + *Key must not be " " string, value can be anything*
  + *It should be stored in .js file*

{

type : phone price: 400

features : ["Camera", "Wifi"]

}

***NOTE*** *:*

* + *Jason.stringify method converts JS object to the string of JSON object(Whole object is a single string)*
  + *Json.parse converts JSON object to a java script object*
* ***Rest Parameter***
  + *is used to store rest of the parameters (remaining parameters) in the form of array in functions*
  + *rest parameter is used to store the rest of the array elements in the form of array in array destructuring*
  + *rest Parameter is used to store rest of the object properties in the form of objects in object de-structure*
  + ***Rest operator will always be at the end/ last parameter***

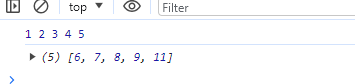
function add(a,b,c,d,e,...z)

{

console.log(a,b,c,d,e) console.log(z)

}add(1,2,3,4,5,6,7,8,9,11)

***Output :***



* ***Destructuring***
  + *It is the process of extracting the array elements and storing them inside individual variables and extracting object properties and storing them inside individual variables*
  + ***There are 2 types of destructuring.***

1. ***Array Destructuring***
2. ***Object Destructuring***

* ***Array Destructuring***
  + *It is the process of extracting array elements and storing them inside a individual variable*
  + *It avoids using array name again and again when accessing array elements*
  + *When we do array re-structuring elements will be stored inside the variables in sequential order*
  + *If we want to skip he elements or if we dont want to store the elements just pass , at the time of de-structuring*

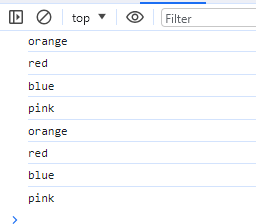
***Example :***

let colors = ["orange", "red","blue", "pink"] console.log(colors[0]) console.log(colors[1]) console.log(colors[2]) console.log(colors[3])

// By using Array Destructure

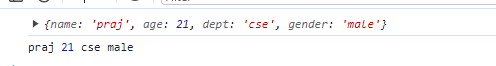
let [o,r,b,p] = colors console.log(o) console.log(r) console.log(b)

console.log(p)

***Output:***

* ***Object Destructuring***
  + *It is the process of extracting object properties and storing them inside a individual variable*
  + *it allows us to avoid using object name again and again when we access object properties*
  + *In this, the variable names must be same as key names*
  + *If we want to skip or dont want to store the property, just ignore the key name at the time of destructuring*

***Example :: Output:***



let student = {name:"praj", age:21, dept:"cse",gender:"male"} console.log(student)

let{name, age, dept,gender} = student; console.log(name, age, dept,gender)

let{name, age, dept}=student;

console.log(name, age, dept)



// DESTRUCTURE USING FUNCTIONS

let student = {name:"praj", age:21, dept:"cse",gender:"male"} let getstudentdata = ({name, age, dept,gender}) =>{ console.log(name, age,dept,gender)

}

getstudentdata(student)

***Output :***



* ***Nested Object destructuring Examples :***

let student =

{

name : "praj", age:24, address :

{

home:{city:"Bangalaore", state:"Karnataka"}, work: {city:"mumbai"}

}

}

console.log(student)

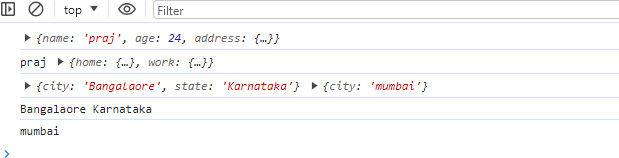
***Normal way >>***

let {name, address} = student let {home,work}=address

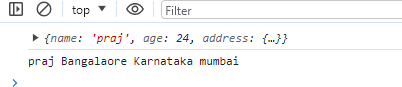
let {city,state}=home let {city:newCity}=work

console.log(name, address) console.log(home,work) console.log(city,state)

console.log(newCity)



***The entire code can be destructured by using the below***



let{name, address:{home:{city,state},work:{city:newcity}}}=student

console.log(name,city,state,newcity)

***Other Functions in Objects:***

***clg(object.keys(student)) // Prints object Key***

***clg(object.values(student)) // Prints object Value clg(object.entries(student)) // Prints both Key and Value clg(object.keys(student.length)) // Prints student length object.seal(student) // can perform only update clg(object.isfrozen(student)) // Cant prform any insert/update/ delete clg(object.isSealed(student)) // Tru or False***

***delete student.age // deletes Students age***

***let newobj = object.create(student) clg(newObj)***

***clg(newObj.sname)***

***let newObj = object.assign(student,employee) clg(newobj)***

***clg(student) // student will be the target where ethe key and the value from the employee will be copied***

***Date***

***d =new date()***

***clg(d.getfullyear()) clg(d.getdate()) clg(d.getmonth()) clg(d.getday())***

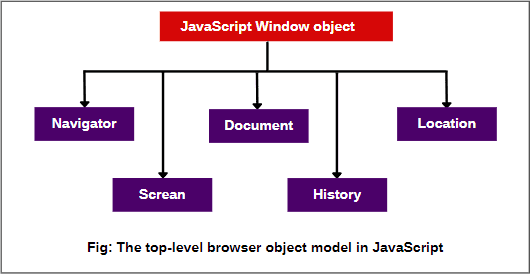
***clg(d.gehours()) clg(d.getminutes()) clg(d.getseconds()) clg(d.tolocaledatestring) clg(d.tolocaletimestring)***

**BOM** stands for Browser Object Model

***BOM***

When we execute the html file in the browser, it acts as an object and the object name is BOM.

It’s a global Object and also known as Window Object



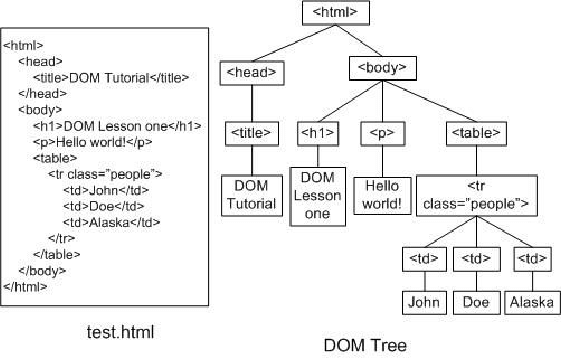
***DOM : DOCUMENT OBJECT MODEL***

* *When we execute the html in the browser, the browser converts the html file into tree like structure and this tree like structure is known as DOM.*
* *Elements section in the browser dev tool represents DOM*
* *DOM is used to manipulate the html through JavaScript*
* *By using DOM, we can create a new HTML elements, we can delete the existing html elements, we can add/update the html elements.*
* *By using DOM, we can create delete update the attributes as well.*
* *By using DOM, we can add, update and delete the CSS as well.*
* *Every tag in the DOM tree is known as node of the tree and every node is an object.*
* *Attributes of the tags are considered as the properties of an object.*
* *Whatever is available on the DOM tree, that will be displayed on the webpage.*
* *In DOM tree, DOM object is a parent Object.*

***DOM METHODS TO GET THE HTML ELEMENTS***

1. ***Documents.getElementbyID(“ID”)***
2. ***Document.getElemntbyClassname(“Classname”)***
3. ***Document.getElemntbyName(“Name”)***
4. ***Document.querySelector(“selector”)***
5. ***Document.querySelectorall(“Selctor”)***

***DOM MODEL DIAGRAM***



1. ***Documents.getElementbyID(“ID”)***
   * *It is used to get the elements based on ID.*
   * *It accepts one arg as string where the arg is elements ID*
   * *If the ID is available for any element, it returns first occurred element*
   * *If the ID is not available then it returns null*

***Example :***

<div id="first\_division">

Let res = document.getElementById("first\_division")

1. ***Document.queryselector(“selector)***

### It gets the element based on the selectors.

* + *It accepts one arg as string where the arg is elements selector*
  + *If the selector is available for any element, it returns first occurred element*
  + *If the selector is not available then it returns null*

## Example :

Let res = document.querySelector("h1")

## Document.queryselectorall(“selectors”)

* + *Same as the query selector but selects all the occurrences*
  + *If the selector is available, it returns it returns the elements in the form of node list*

## Document.getelementbyclassname(“classname”)

### It is used to get the elements based on the class name.

* + *It accepts one arg as string where the arg is classname*
  + *If the selector is available, it returns it returns the elements in the form array like object else empty html collection.*

## Document.getelementbytagname(“element”)

### It is used to get the elements based on the tag name.

* + *It accepts one arg as string where the arg is tagname*
  + *If the selector is available, it returns it returns the elements in the form of html collection or empty html collection*

## DOM METHODS TO GET THE ATTRIBUTES

### Let input = document.queryselctor(“input”) Clg(input)

*Clg(input.type) Clg(input.minlength)*

## ADDING CSS THROUGH JS

* + ***Let a = Documents.getElementbyID(“abc”)***

## // abc.style.color = “red”;

***Event Handlers in JavaScript***

# Hoisting

* *It is the process of moving the variable declaration and function declaration to the top of their scopes*
* *There are 2 types of hoisting*
  + *Function hoisting*
  + *Variable hoisting*
  + *Function hoisting*

1. *It is a process of moving function declaration to the top of its scope before execution of the program*
2. *It is very useful and is used in many scenarios*

# Example :

Clg(m1) Function m1()

{

Clg(“m1”)

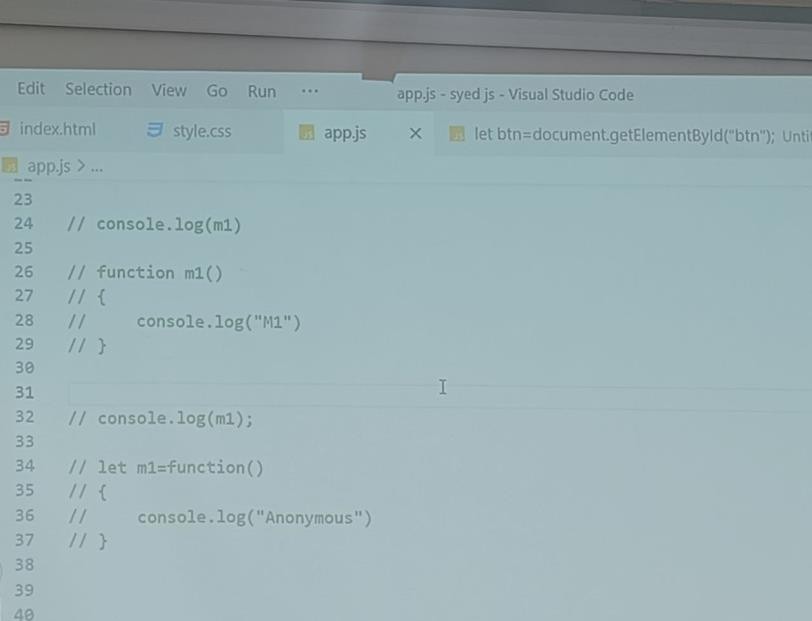
}

Clg(m1) Output :

M1 M1

##### Note:

1. We can call the function before the function declaration. That’s the major Advantage.
2. Only Named functions will be hoisted
3. Only var keyword is used for the Hoisting and doesn’t support let and const.

***Examples with different scenarios:***

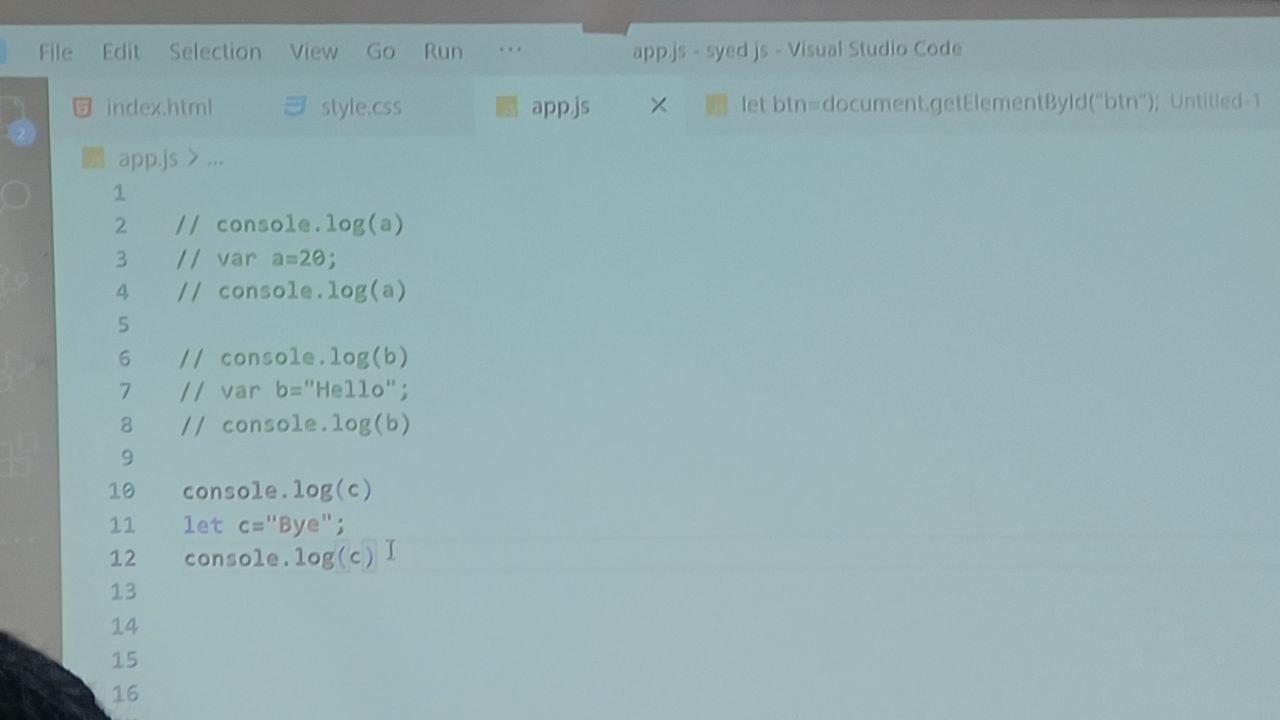
##### Variable hoisting

1. *It’s the process of moving variable declaration to the top of its scope before the*

execution of the program.

1. *Only var type of variables can be hoisted*
2. *let and const type of variables can’t be hoisted*
3. *If we try to access the variables of type const and let before its declaration we get error*
4. *If we try to access the variables of type var, before its declaration, we get*

***undefined*** *.*

***Examples with the different Scenarios***

***Output::***

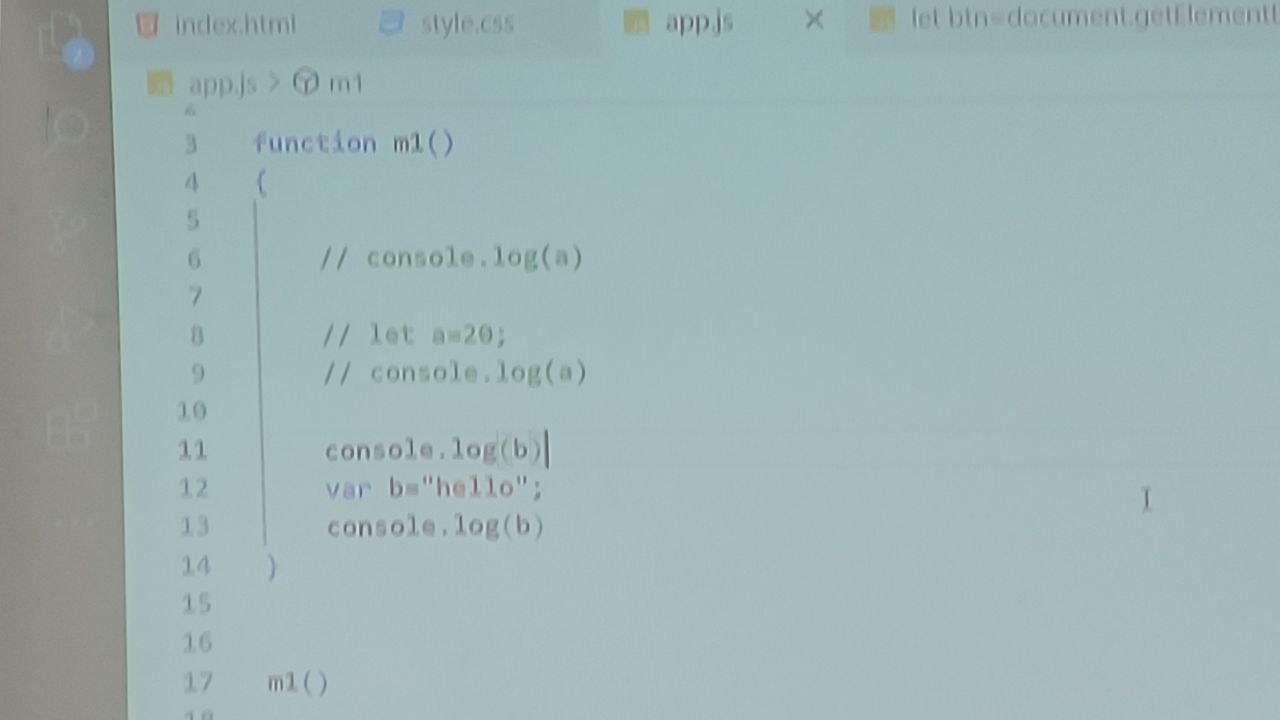
* 1. ***Undefined 4. 20***

***6.Undefined***

***8. “Hello”***

1. ***Error***

***12. “Bye”***

***Output::***

***6. Error***

***9. 20***

1. ***Undefined***

***13. Hello***

***To make JS Asynchronous, we***

***use call back functions, promises, await, asysnc***

***Callback : Function as an argumrnt Promises : Object***

***await and async : Keywords***

***Callback Function ::***

***It is function which is passed as an argument to another function It is used to handle asynchronous type of data.***

***We dont prefer callback function as it creates callback hell which is vry difficult to maintain and understand.***

***Callback hell is nothing but the nestd callbacks***

***Promises::***

***Its a built in objct in JS which is usd to handle asynchronous type of data. Promise contructor accepts one argunemt whre the arg is a callback function The callback function accepts 2 args, ]***

***>> 1st arg is rsolvd function***

***>>2 is rject functiom***

***The rsolved funct will be executed when the promiss is fulfilled The rjectd funct will be executed when the promiss is rejcted***

***The Promises will hav 3 stats:***

1. ***Fulfilld or rsolved***
2. ***Pnding***
3. ***Rejected***

***IF we want to handle rsolved or fulfilld data then we use "then-block" or "then method" If we want to handle rejected data we use "catch- block" or catch method***

***When we fecth the apIs by using fecth () of JS or any other library, then we get promise objct as response***

***The rspone can either be resolved/pending or rejected state***

***Pending >> thn if fulfilled lse rejectd(catch()) >> back to pending***

***let a = 60;***

***let p = new Promise((resolve, reject)=>***

***{***

***if (a===40)***

***age:"25" })***

***}***

***else***

***{***

***}***

***)***

***{***

***resolve({name:"praj",***

***rejected("Unsuccesful")***

***Inorder to9 accss th rsolvd data, we use then i.e,***

***For rejcted, we us catch mathod***

***p.catch((error)=>***

***{***

***clg(error)***

***}***

***)***

***For rsolvd. w use then() p.then((data)=>***

***{***

***clg(data) clg(data.name.toupprcase())***

***}***

***)***

***Note :: We can pass multiple then blocks but not the multiple catch blocks.***

***th rturn value of the 1st then block will be snt as the argument to th 2nd then block functioon and so on..***

***Example >>***

***let p = new Promise((resolve, reject)=>***

***{***

***if (a===40)***

***{***

***age:"25" })***

***}***

***else***

***{***

***}***

***)***

***resolve({name:"praj",***

***rejected("Unsuccesful")***

***p.then((data)=>***

***{***

***clg(data) clg(data.name.touppercase())***

***return json.stringify(data)***

***}***

***)***

***p.then((d)=>***

***{***

***clg(d) // stors the return value of th 1st then block***

***}***

***)***

***Finally >> if a st of code to b performed whether it is rsolved or rejctd, w us finally block finally(()=>***

***{***

***clg("Finally am done") // this block will execute whatver the output comes***

***})***

* ***Async and Await***
  + *These are the keywords in JavaScript which is used to handle Asynchronous type of data.*
  + *Async keyword makes the function Asynchronous.*
  + *Async function return Promise.*
  + *Await keyword must be inside in Async function*
  + *Await keyword await only for the promise But it allows other statements or tasks to execute*
  + *If you want to handle the errors, we use try catch block, where try block handles the resolved data and catch block handles the rejected*
  + *Async and await are syntactic sugar for promises*

***Example by using api>>***

**Aysnc function gtdata()**

**{**

**Try**

**{**

**Let users = await fetch(“api.github.com/users”) Users = await users.json() Users.map({avatar\_url})=>**

**{**

**Document.body.innerhtml += ` <img src = ${avatar\_url} hight weight>**

**})**

**}**

**Catch (err)**

**{**

**Clg(err.messge)**

**}**

**}getdata()**

**REGULAR EXPRESSION**

Its is a built-in object in JavaScript which is used match the patterns. Syntax :

*Bad method : let reg = new RegExp(Pattern)* ***Best method : let reg = /pattern / modifier Patterns :***

***Entering value should value have/match atleast one of the value***

***[a-z] [a-z 0-9]***

***[0-9] [A-Z]***

***[A-Z 0-9] [A-Z a-z 0-9]***

***^ Starts with or Negation***

***$ ends with***

***/ one or more occurrence***

***\* zero or more occurrence***

***Modifiers are below***

***i >> case insensitive first match gi >> case insensitive all match***

***g >> case sensitive all match***