JavaScript

1) Introduction
2) History
3) Key Features
4) JavaScript Engine
5) Ways of Adding JavaScript
6) Comments
7) Tokens
8) DataTypes
9) Scope
10) Hoisting & Temporal Dead Zone
11) Difference b/w var, let, const
12) Operators
13) Decision Statements
14) Loops
15) Functions
16) Currying
17) Execution Context
18) Strings
19) Arrays
20) Objects
21) Math & Date Object
22) For-In & For-Of loop

23) Spread operator & Rest Parameter

24) SetTimeout & SetInterval

25) BOM (Browser Object Model)

26) DOM (Document Object Model)

27) Events and its Propogation

28) Promises.

29) Async & await

30) Fetch Method

31) Form Validation

1) Introduction

→ JavaScript is one of the most popular programming language in the World.

→ It is not only used as a Frontent Technology but also as a Backend Technology.

Frontent → React JS

Backend → NodeJS, Express JS

*→ JavaScript is a Scripting Language that is used to add functionalities to our Webpages.

2) History

→ Earlier in the year 1990's a Company named Netscape wanted to build a Scripting Language for their Browser called Netscape Navigator.

→ They hired a person called Bredan Eich and asked him to develop a Scripting Language.

→ He developed within 10 days and named it as Mocha in the year 1995

→ Later it was named as LiveScript but as marketing strategy they change its named to JavaScript

→ To make a JavaScript as Open Source they handed it to ECMA (European Computer Manufacturing Association) in the year 1997. So it is called as ECMA script.

→ To run JS code outside the browser a person named Ryan Dahl introduced NodeJS

3) Key Features

→ JavaScript is a Scripting Language because

i) Each and every code written in JavaScript can be read, analyxe and executed in the browser

→ Java Script is a Weakly / Loosely typed language

i) Even if the semicolon (;) is missing or the DataType is not specified it does not provide any error

Eg :-

Java	JavaScript
int a=10;	a=10 → Datatype
S.O.Pln (a);	Clg (a) → semicolon

→ JavaScript is a Dynamically typed language

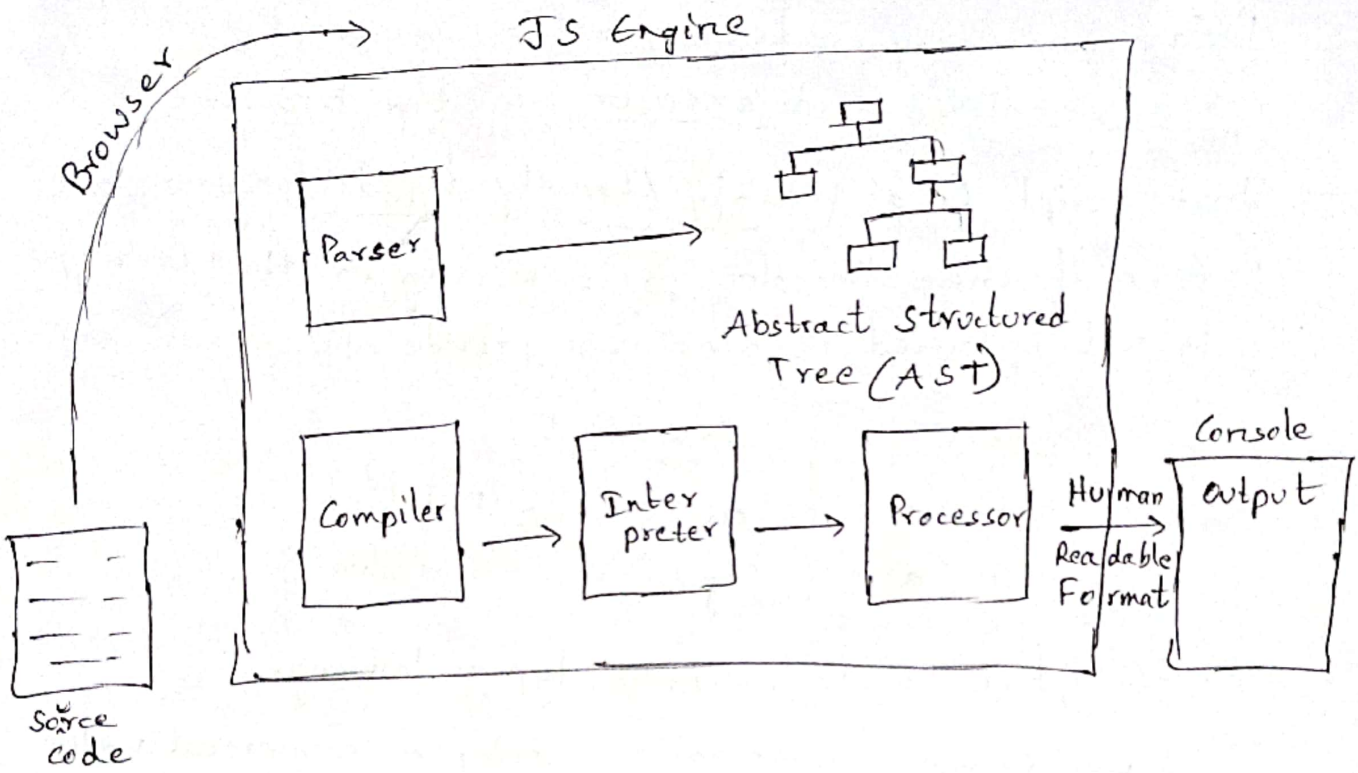
i) If we declare a variable storing a numerical value can be later reassigned with any other type of value

Eg :-

Java	JavaScript
int a=10;	a=10
it is not possible ← a=true	a=true → it is possible

→ JavaScript is a Synchronous single Threaded / Interpreted language because it reads the code line by line and executes, if there any errors it will stop the execution without moving to the next line.

4.) JavaScript Engine



Browser	JS Engine
chrome	V8 Engine
Firefox	Spider Monkey
Internet Explorer	Chakra
Brave	Blink+V8

→ When the JavaScript file (source code) is executed on the browser, it is sent to the JS Engine.

→ In this process the source code is first send to the Parser, which stored the original code into tree like structure known as Abstract Structured Tree (AST) and it also devides our code into parts (Tokens) and sends one by one to the compiler

→ The compiler will check for the syntax errors and converts the source code into binary code (0's & 1's) and sends it to the Interpreter.

→ The Interpreter checks the Binary code line by line and sends into the Processor.

→ The processor will convert the binary code into Human Readable Format and prints the output in the Browser's Console

5) Ways of Adding JS

There are 2 ways of adding JS, they are

1) Internal JavaScript
2) External JavaScript

1) Internal JS :

→ It refers to the writting of the JS code inside the body tag by maintaining the script tag

Eg :

```
<body>
    <script>
        console.log("I am Internal JS")
    </script>
</body>
```

2) External JS :

→ It refers to writting of the JS code by maintaining a seperate file saved with the extension .JS

→ To link JS and HTML file we use script tag along with src attribute which takes in the path of JS file

Eg :- .js

```
<body>
  <script src="Path of JS file"> </script>
</body>
```

6) Comments

→ Comments are the piece of code that is not executed on the Browser.

→ It is mainly used to improve the Readability of the code

Types of Comments :- There are 2 types

1) Single line comment (//)
2) Multiline comment (/* --- */)

7) Tokens

→ Tokens are the smallest parts of any programming language

Types of Tokens :-

Tokens are divided into 3 categories

1) Keywords
2) Identifiers
3) Literals

1) Keywords :

→ These are pre-defined or reserved words.

→ These should always be written in lower case

Eg: console, log, var, let, const, if, else, for - - -

2) Identifiers :

→ These are the names given for the components like variables and functions.

Rules :

i) Identifiers should not be same as keyword

ii) It should not start with a number but it can contain a number.

iii) Except '$' & '_' no other special characters should ~~not~~ be used.

Eg: skills, name, age - - -

3) Literals :

→ These are the values that are stored in a variable

Eg: 10, true, false, "JavaScript" - - -

8) Datatypes

→ It tells us the type of data stored in a variable

Types of Datatypes

Datatypes are divided into two categories:

1) Primitive Datatypes
   1) Number
   2) String
   3) Boolean
   4) Undefined
   5) Null
   6) BigInt
   7) Symbol

2) Non-Primitive Datatypes
   1) Object

1) Number :- It represents both integer and decimal values

Eg :-
```
var num1 = 10
var num2 = 10.25
var num3 = 1000000.56
console.log(num3) // 1000000.56 → number
console.log(num2) // 10.25 → number.
```

2) String :- It represents a character or the sequence of characters enclosed in ' ', " ", ` `

Eg :-
```
let str1 = 'JavaScript'
let str2 = "R"
let str3 = `10`
console.log(str3) // 10 → string
console.log(str1) // JavaScript → string
console.log(typeof str2) // string
```

3) Boolean :- It represents a logical value either true or false

Eg :- let bool1 = true
let bool2 = false
console.log(bool2) //false → boolean
console.log(bool1) //true → boolean

*4) Undefined :- It refers to uninitialize variable or absence of a value for a variable

Eg :- let val
console.log(val) //undefined
console.log(typeof val) //undefined

*5) Null :- It is the intentional absence of a value for a variable

Eg :- let name=null
console.log(name) //null
console.log(typeof name) //object
name = "Banto"
console.log(name) //Banto
console.log(typeof name) // string.

6) BigInt :- This datatype is used to store a large number without loosing the precision.
→ To convert number to a bigInt value specify 'n' at the end of a number.

Eg:-

```
let bigIntNumber = 12345678912345678n
console.log(bigIntNumber) // 12345678912345678n
console.log(typeof bigIntNumber) // bigInt
```

7) Symbol :- This DataType is used to generate unique id's for an object

Eg:-
```
let sym = Symbol(1)
console.log(sym) // symbol(1)
console.log(typeof sym) // symbol
```

9) Scopes

It is the region in which the variable are accessible

Types of scopes :-
There are 3 types of scopes in Javascript

1) Global scope
2) Local/function scope
3) Block scope

1) Global scope :- Variables declared at the top level should we accessible in the same Javascript file.
→ Variables declared with var keyword follows global scope where as w.r.to let and const keywords follows script scope.

Note :- script scope is similar to global scope which was introduced along with the creation of let and const keywords in ~~ES~~ script ES6 version

and JS released in 2015.

2.) Local / Function Scope : Variables declared inside the function should be accessible only inside the function.
→ Variables declared with var, let, const follows local / function scope

3.) Block Scope : Variables declared inside the block should be accessible only inside block.
→ Variables declared with let and const keywords follows block scope whereas variables declared with var keyword does not follow block scope

	Global	Script	Local	Block
var	✓	×	✓	×
let	×	✓	✓	✓
const	×	✓	✓	✓

10) Hoisting

It is the process of moving the variable declarations to the top of the scope

(or)

It is the process of accessing the variables even before its declaration

→ Hoisting is completely dependent on Memory allocation but not the output

→ If memory allocation is done for a variable then hoisting is done else hoisting is not done.

→ Variables declared with var, let, const keywords are hoisted and the values are initialized as follows.

var → undefined

let → value unavailable

const → value unavailable

Note :- Values are not initialized for let and const because of Temporal Dead Zone.

* Temporal Dead Zone :- (TDZ)

It is the time duration between the creation of a variable and its utilization

Eg:-
```
console.log(a); // undefined
var a=10;
console.log(b); //uncaught Ref Error: Cannot access 'b' before initialization
let b=20;
console.log(c); //  "  "  "
const c=30;
```

11) Difference between var, let and const

Var	let	const
Scopes - global, local	Scopes - script, local, block	Scopes - script, local, block
Declaration without initialization is possible Eg: var a; console.log(a);	Declaration without initialization is possible. Eg: let b; console.log(b);	Declaration without initialization is not possible. Eg: const c; console.log(c);

Re-assigning the variable is possible
Eg: var a=10
a=40;
console.log(a);
	Reassigning the variable is possible
	Eg: let b=10;
	b=true;
	console.log(b);
		Re-assigning the variable is not possible
		Eg: const c=30;
		c=false;
		console.log(c);

Redecloration of the variable is possible
Eg: var a=10;
var a=25;
console.log(a);
	Re-declaration of the variable is not possible
	Eg: let b=20;
	let b=30;
	console.log(b);
		Redeclaration of the variable is not possible
		Eg: const c=35;
		const c=40;
		console.log(c);

Hoisting is done & value is initialized as undefined
Eg:
console.log(a);
var a=10;
	Hoisting is done & value isn't initialized because of TDZ
	Eg:
	console.log(b);
	let b=20;
		Hoisting is done & value isn't initialized because of TDZ
		Eg:
		console.log(c);
		const c=30;

12) Operators

→ Operators are the predefined symbols used to perform a specific task.

→ Operands are the values upon which the operation is performed

→ Combinations of operators & operands is known as Expression

Types of Operators :- 5 types

1) Arithmatic Operators (+, -, *, /, %)

2) Comparision Operators (>, <, >=, <=, !=, ==, ===)

3) Logical Operator (||, &&, !)

4) Assignment operator (=, +=, -=, *=, /=, %=)

5) Conditional / Ternary operator.

1) Arithmatic Operators

Addition

```
console.log (2 + 3) //5
console.log (typeof (2+3)) // ~~string~~ number
console.log (2+"3") //23
console.log (typeof (2+"3")) //string
console.log ("2"+"3") //23
console.log (2+true) //3
console.log (2+false) //2
```

```
console.log (2+"2"+false)
                    //22 false
dlg (2+5+"2"+true) //72true
dlg (2+10+'abc') //12abc
```

Subtraction

```
clg(2-3) //-1
clg(typeof(2-3)) // number
clg(2-"3") //-1
clg(typeof(2-"3")) //number
clg("2"-"3") //-1
clg(2-true) //1
clg(2-false) //2
clg(2-"2"-false) //0
clg(2-5-"2"-true) //-6
clg(2-10-'abc') //NaN
```

↓ Not a Number

Multiplication

```
clg(2*3) //6
clg(typeof(2*3)) //number
clg(2*"3") //6
clg(typeof(2*"3") //number
clg("2"*"3") //6
clg(2*true) //2
clg(2*false) //0
clg(2*"2"*false) //0
clg(2*5*"2"*true) //20
clg(2*10*'abc') //NaN
```

Division

```
clg(4/2) //2
clg(typeof(4/2)) //number
clg(4/"2") //2
clg(typeof(4/"2")) //number
clg("10"/"5") //2
clg(2/true) //2
clg(2/false) //Infinity
clg(2/"2"/false) //Infinity
clg(10/5/"1"/true) //2
clg(2/10/'abc') //NaN
```

↓ Not a Number

Modulus

```
clg(4%2) //0
clg(typeof(4%2)) //number
clg(4%"2") //0
clg(typeof(4%"2") //number
clg("10"%"5") //0
clg(2%'abc') //NaN
```

2) Comparision / Relational Operator

```
let p=1;
let q=2;
let r="2";
clg(p>q); // false
clg(p>r); //false
clg(p<r); // true
clg(p>=r); // true
clg(q<=p); //false
clg(p!=q); // true
clg(r!=q); //false
clg(p==r); //false
clg(q==r); // true
clg(q===r); //false
```

3) Logical Operator

```
let a=1
let b=2
let c="2"
clg(a>b||b>c) //false
clg(a>b||b==c) //true
clg(a>b && b>c) //false
clg(a>b && b==c) //false
clg(a<b && b==c) // true
clg(!(a==b)) //true
clg(!(b==c)) //false
```

4) Assignment Operator

```
let a=20;
let b="10";
clg(a+=b) //2010
clg(a-=b) //2000
clg(a*=b) //20000
clg(a/=b) //2000
clg(a%=b) //0
```

5) Conditional / Ternary Operator

Eg:1

```
let pellicheskuntava= false;
pellicheskuntava? clg("yes"):
                  clg("No") // No
```

Eg:2

```
let weekend = false;
weekend? clg("yes"):clg(
                  "No");
                  //No
```

Note :-

NaN :- It stands for Not a Number which is used to represent a invalid mathematical operation

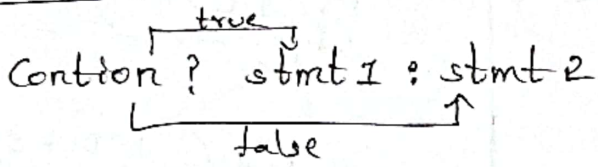
Eg :- clg(2/10/'abc') //NaN

Difference b/w == & ===

==	===

→ It is used to compare only the values
Eg :- clg(1=="1") //true
clg(2==1) //false
	→ It is used to compare both the values and the Datatype
	Eg :- clg(1==="1") //false

→ Syntax :- Conditional operator



---

13) Decision statements

They help to execute the statements based on the conditions

Types of Decision statements :-

1) If statements
2) If else statements
3) Else if
4) Switch case

1) if statements :

Syntax :

```
if(condition)
{
  //block of code is to be
    executed if condition is
    true
}
```

Eg :

```
let area="Ibnagar";
if(area==="Dilsukhnagar")
{
  clg("Hi Guys");
}
```

2) if else :

Syntax :

```
if(condition)
{
  //if condition is true
}
else
{
  //if condition is false
}
```

Eg :

```
let amount=50;
if(amount>=350)
{
  clg("PISTA HOUSE");
}
else if(amount>200 &&
        amount<=300)
{
  clg("RAGHAVENDRA");
}
else
{
  clg("PADUKO");
}
```

4) Switch case :

Syntax :

```
switch(condition)
{
  case value 1: Code block;
                break;
  case value 2: Code block;
                break;
        !
        !
  default: code block;
}
```

Eg :

```
let trainer="Pavan";
switch(trainer)
{
  case "Abbas":
    clg("Programmer");
    break;
  case "Pavan":
    clg("Java")
    break;
  case "Yasin":
    clg("SQL");
  default:
    clg("Invalid");
}
```

3) else if :-

Syntax :-

```
if (condition 1)
{
  // if condition is
     true
} else if (condition 2)
  {
  // condition 2 is
     true
  }
     .
     .
     .
  else
  {
    //default
  }
```

Eg :-

```
let amount=50;
if (amount >=350)
{ clg ("Pista House")
}.
else if (amount >200 && amount
                    <=300)
 { clg ("Raghavendra")
  }
 else
 {
 clg ("Paduko")
 }
```

Example for switch case :-

```
let amount =300;
switch (true) {
 case (amount >=350) : Console.log ("PistaHouse");
     break;
 Case (amount >200) && amount <=300) : console.log
                                ("Raghavendra");
     break;
 Case (amount >100) && amount <=200) : console.log
                                ("outside");
     break;
 default : console.log ("Paduko");
}
```

14) Loops

→ Loops are used to perform a task repeatedly 'n' no.of times based on a condition

Types of loops :

1) for loop
2) while loop
3) do while loop

1) for loop :

Syntax : for(①initialization ; ②Condition ; ④Updation)
{
③ // Code
}

Eg :
```
for(let i=1 ; i<=5 ; i++)
{
  document.writeln("<h1> Happy Birthday </h1>")
}
```

2) while loop :

Syntax :
```
Initialization ①
while(Condition) ②
{
  //Code ③
  Updation ④
}
```

Eg :
```
let j=1;
while (j<=10) {
  document.writeln("<h1> SOMETHING </h1>")
} j++
```

3) do while loop :-

Syntax :-

```
initialization ①
do {
    //code ②
    updation ③
} while(condition) ④
```

Eg :-

```
let k=1;
do {
    document.writeln("<h1>ANYTHING</h1>")
    k++
} while (k<=0)
```

Note :-

The difference between while loop and do-while loop is that while loop will only execute if the condition is satisfied whereas do while loop will execute atleast once even if the condition is not satisfied.

15) Functions

→ Functions are the block of code that is used to perform a specific task

→ To execute the function, we need to call the function

Advantages :-

1) It improves the readability of the code
2) It increases the reusability of the code by calling it multiple times.

Types of functions

In general there are 2 types of function in JavaScript

1) Function Declaration :- → declaration
   ① Normal / General / Common function.

2) Function Expression :-
   ① Anonymous function
   ② Arrow function
   ③ Immediate Invoked Function Expression (IIFE)

1) Function Declaration

1) Normal / General / common function :-

Syntax :-

function declaration part ←

```
function identifier (parameters)
{
    //code
}
```

function calling statement ← `identifier (arguments)`

Eg :-

```
//normal/General/common fn
function anything ()
{
  let a=10;
  let b =20;
  console.log (a+b);
}
anything ();
anything () ;
```

o/p :-

30
30

//Behavior of function

//Function without parameter without return keyword

```
function greet ()
{
  clg ("Good Evening");
}
greet ();
```

o/p :-

Good Evening

//Function without parameter with Return keyword

```
function expressYourFeelings ()
{
  return "I Love You".
}
clg (expressYourFeelings ())
```

o/p :-

I Love You

//Function with Parameter without Return keyword

```
function add (a, b)
{ clg (a+b)
}
add (10, 20)
add (20, 30)
```

o/p :- 30
50

// Function with parameter with return keyword

```
function mul(a,b)
{
  return a*b
}
clg(mul(2,3))
clg(mul(30,10))
```

o/p :- 6
300

// ! Extra Parameters

```
function extraParam(a,b,c)
{
  clg(a+b+c) //NaN
}
extraParam(1,2)
```

o/p :- NaN

// ! Extra Arguments

```
function extraArgs(a,b)
{
  clg(a+b) //3
}
extraArgs(1,2,3)
```

o/p :- 3

2) Function Expression

→ It provides way to create the function without specifying the identifier (name of the function)

→ Generally these type of functions need to be stored in a variable to execute

① Anonymous function :-

These are the function that does not have any name for it

Syntax :- variable = function(parameter)
{
// code
}
variable(argument)

// 1) Anonymous function

Eg :- 1

```
let anonymous1 = function()
{
  clg("I am anonymous function")
}
anonymous1()
```

o/p :-
I am anonymous function

Eg :- 2

```
let anonymous2 = function(val1, val2)
{
  return val1-val2
}
clg(anonymous2(20,10))
```

o/p :-
10

② Arrow function :-

→ It is a function that does not have function keyword as well as identifier

→ It is a type of function introduced in ES6 version of javascript

Syntax :-

```
variable = (parameters) =>
    {
      // code
    }
variable(arguments)
```

Eg :-

```
let greet = (msg) =>
  {
    console.log(msg)
  }
greet("Good Evening")
```

Types of Arrow Function :-

Based on the Return keyword arrow function is devided into two types.

1) Explicit Return arrow function
2) Implicit Return arrow function

1) Explicit Return arrow function :-

It is a type of arrow function that involves the use of Return keyword wantedly

Eg :-

```
let product1 = (a, b) => 
  {
    return a*b
  }
console.log (product1(10,10))
```

O/p :- 100

2) Implicit Return arrow function :-

It say that if there is only one line of statement to be executed then we can skip the { } braces as well as Return keyword

Eg 1 :- Eg 1

```
let product2 = (a,b) => a*b          O/P :- 100
console.log (product2(10,10))
```

Eg 2 :-

```
let add1 = (a,b) => console.log (a+b)   O/P :- 20
add1(10,10)
```

Behaviour of Arrow function :-

1) Single Parameter :- Whenever there is single parameter in the arrow function we can skip the paranthesis() for it

Eg :-
```
let singleParam = msg =>
{
  console.log (msg)
}
singleParam("Hello")
```

2) No Parameter :- Whenever there is no parameter in the arrow function we can skip the paranthesis() and replace it with underscore _.

Eg :-
```
let info = _ =
{
  console.log ("JavaScript classes started")
}
info()
```

3) Immediate Invoked function Expression (IIFE):-

→ These are the functions that should be executed immediately after its declaration

→ These functions will execute only once

→ These functions can be normal or Anonymous or Arrow, but should be wrapped inside the paranthesis ( )

Syntax:-

```
(
  Normal/Anonymous/Arrow
)();
```

Eg:- Normal

```
(function normal(){
  console.log("Normal function")
})
();
```

Eg:- Anonymous

```
(function ~~error~~(){
  console.log("Anonymous Function")
})
();
```

Eg:- Arrow

```
(()=>{
  console.log("Arrow function")
})();
```

Higher Order Function (HOF)

→ Higher order functions are those functions that take another function as its argument and return a new function.

→ Callback function are the values that are being passed as an argument to HOF

Eg :-

```
let add=(a,b)=>
  {
   return a+b
  }
  let sub=(a,b)=>{
     return a-b
  }
  let mul=(a,b)=>{
     return a*b
  }
  let calculator=(a,b,task)=>{
   return task(a,b)
  }
  console.log(calculator(20,20,add)//40
  consolelog(calculator(40,20,sub)//20
  console.log(calculator(10,5,mul)//50
```

Note :-

In the above example calculator acts like a higher order function and the functions like add, sub, mul acts like the callback function.

Hoisting in Functions

→ Hoisting is the process of accessing or calling the function even before its declaration.

→ Hoisting is completely dependent on Memory allocation but not the output

→ If memory allocation is done for a function the hoisting is done else hoisting is not done.

→ Functions like normal, Anonymous and arrow are hoisted

→ But the values in function expression completly depend on the keywords like var, let and const

	Value
normal	Entire function
anonymous → var	undefined
arrow → let, const	value unavailable

Eg:-

```
//NORMAL
normal()  //I AM NORMAL FUNCTION
function normal() {
  console.log("I AM NORMAL FUNCTION")
}

//ANONYMOUS
anonymous()  // Uncaught TypeError: anonymous is not func
var anonymous=function() {
  clg("I AM ANONYMOUS FUNCTION")
}
//ARROW
arrow()  // Uncaught ReferenceError: Cannot access 'arrow' before initializ
let arrow=() => {
  clg("I AM ARROW FUNCTION")
}
```

☆ 16) Function Currying

It is the process of transforming a function taking multiple arguments into nested series of function taking single Argument

function taking multiple arguments

↓

```
function add1(a,b,c)
{
  return a+b+c;
}
console.log(add1(10,20,30));
```

nested series of fn taking single argument

```
function add2(a)
{
  return function(b)
  {
    return function(c)
    {
      return a+b+c;
    }
  }
}
clg(add2(10)(20)(30));
```

implicit return arrow function

```
let add3=(a)=>(b)=>(c)=>a+b+c;
clg(add3(10)(20)(30));
```

17) Execution Context

→ It is a area or the environment that holds the information about the memory allocation for the variables and functions, regarding the scope, execution statements etc.

→ JavaScript code will run twice in which it undergoes the variable phase followed by execution phase.

Eg :-

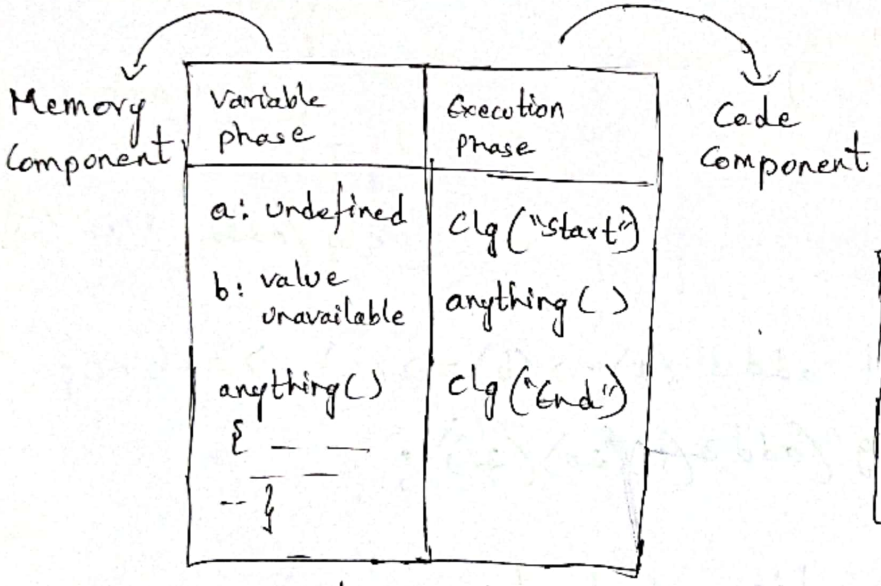
```
console.log("start")
var a=10;
let b=20;
function anything()
{
  const c=30;
  console.log(a+b+c);
}
anything()
console.log("End");
```

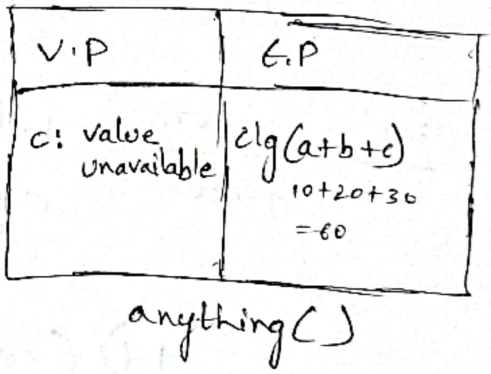
Variable Phase

→ memory allocation for variables & functions

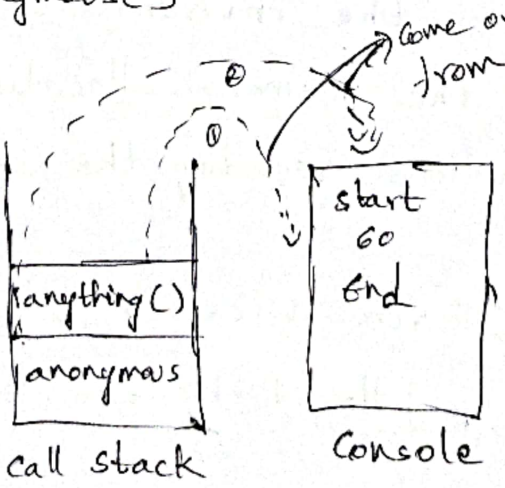
Execution Phase

→ Values will be stored into the variables & other execution statements





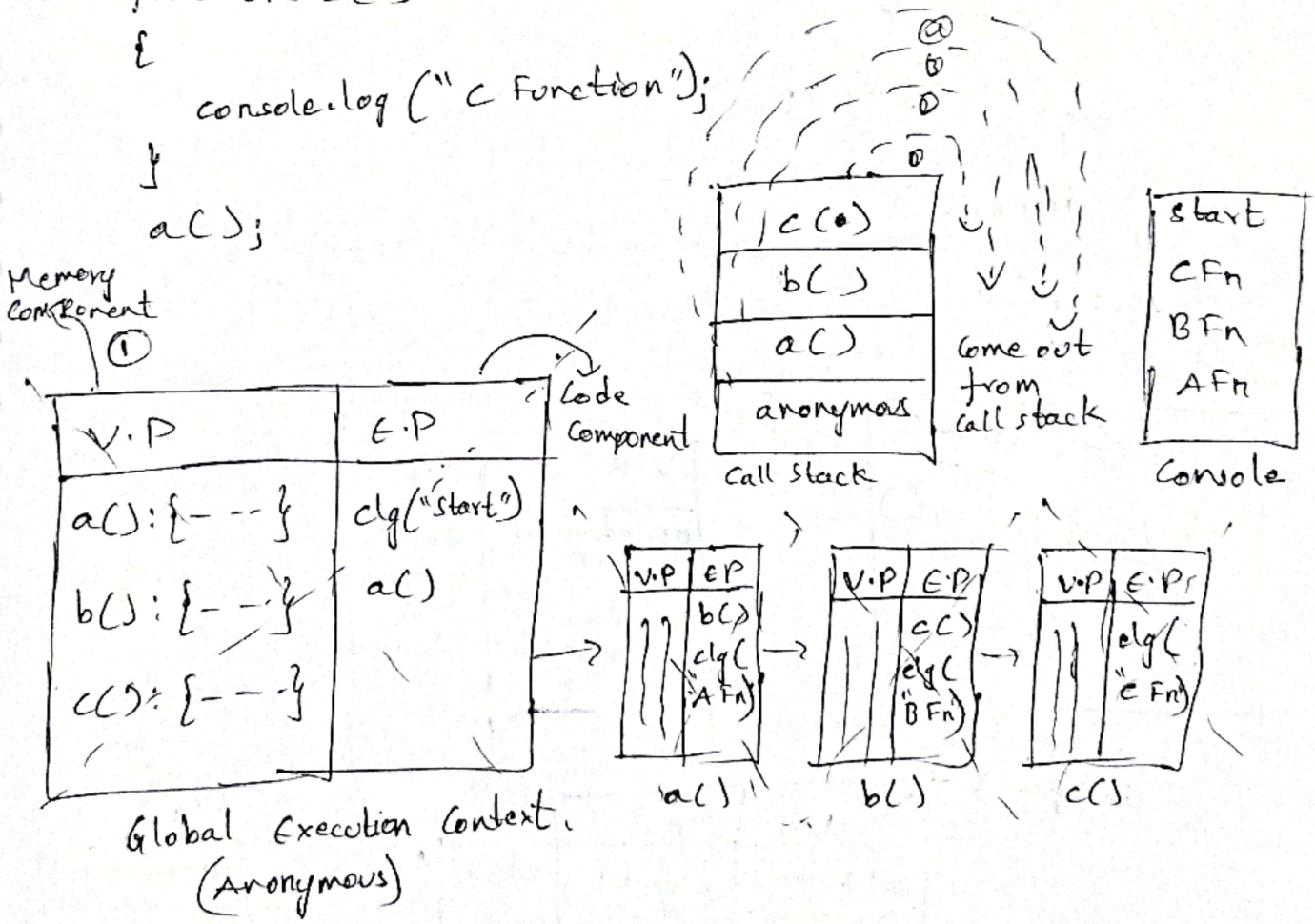
anonymous()



Eg:-2 Execution Context

Prog

```
console.log("start")
function a()
{
    b();
    console.log("A Function");
}
function b()
{
    c();
    console.log("B Function");
}
function c()
{
    console.log("c Function");
}
a();
```



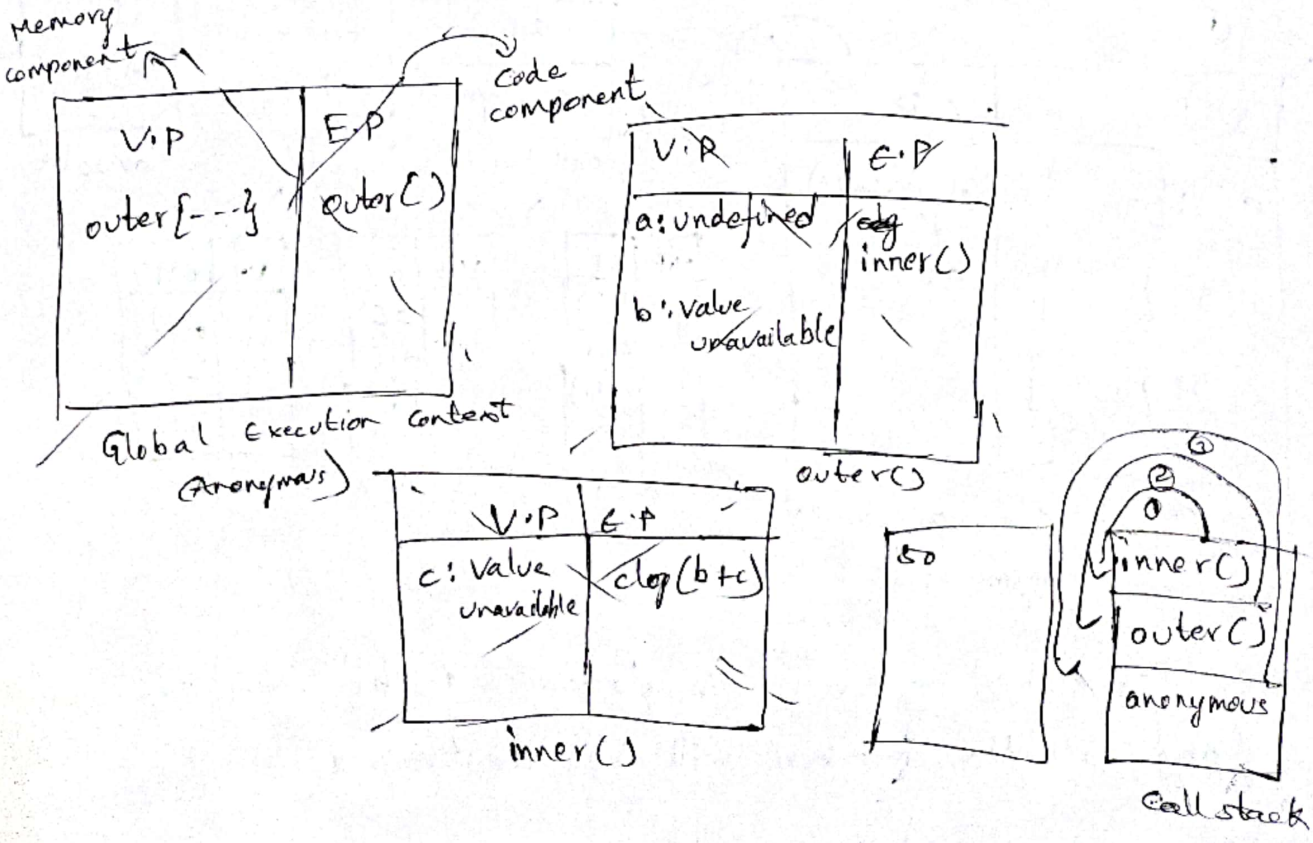
→ After the execution completed Global Execution Context and all the members will be destroyed.

Closures

Closures are the functions that stores the value of a outer function that is required by the inner function even after the completion of the execution of outer function.

Eg:

```
function outer()
{
    var a=10;
    let b=20;
    function inner()
    {
        const c=30;
        clg(b+c);
    }
    inner()
}
outer()
```



18) Strings

→ It is a character or the sequence of characters enclosed in ' ' (or) " " (or) ` ` (backticks)

Ways of Creating a String

There are 3 ways of creating a string in JavaScript

1) Literal Way (' ' (or) " ")
2) Constructor Object
3) String Interpolation (or) Template Strings

1) Literal Way :- It involves the creation of the string using the symbols ' ' (or) " "

Eg :-
```
let str1 = 'JavaScript'
let str2 = "Hyderabad"
clg(str2) // Hyderabad
clg(str1) // JavaScript
```

2) Constructor Object :- It involves the creation of the string using new keyword followed by string constructor.

Eg :-
```
let str3 = new String ("Something")
clg(str3) //String {'Something'}
clg(typeof str3) //object
```

3) String Interpolation (or) Template strings :

It involves the creation of a string using `` (backticks) which helps to dynamically render the variables inside the string using a symbol # $ and { }

Eg:- let firstName = 'Virat'
let lastName = 'Kohli'
let age = 37
clg(`My name is ${firstName} ${lastName} and my age is ${age}`)

O/P :- My name is Virat Kohli and my age is 37

Accessing the characters of the String

→ To access the characters of a string we go with the concept of indexing.

→ Index always start with 0.

```
let str = "BIRYANI"
          0 1 2 3 4 5 6
clg(str[1]) // I
clg(str[5]) // N
```

Note :- Specifying the greater index value than the existing one or if the index value is negative it return undefined

```
clg(str[8]) // undefined
clg(str[-1]) // undefined
```

Note 2 :- To find the total number of characters present in the given string we use length property

Eg:- console.log(str.length) // 7

Methods :-

1) charAt(index) :- This method returns a character present at the specified index

Note :- The specifying the greater index value than the existing one or if the index value is negative it return "" ⟶ Empty string (Blank)

Eg :-
```
let str1 = "MOBILE"
            0 1 2 3 4 5
clg(str1.charAt(4)) // L
clg(str1.charAt(6)) // "" (Empty string)
clg(str1.charAt(-1)) // "" (Empty string)
```

2) charCodeAt(index) :- This method returns the ASCII value for the character present at the given index

Eg :-
```
let str2 = "abc"
clg(str2.charCodeAt(0)) // 97 → ASCII value of a
clg(str2.charCodeAt(2)) // 99 → ASCII value of c
```

~~3) char~~

3) toUpperCase() and toLowerCase() :-

These methods converts the given characters of a string to upper case and lower case respectively

Eg :- let str3 = "SoMeTHInG"

clg(str3.toUpperCase()) // SOMETHING

clg(str3.toLowerCase()) // something

4) repeat(count) :- This method repeats the given string 'n' no.of times base on the count value

Eg :- let str4 = "chai"

clg(str4.repeat(3)) // chai chai chai

5) concat(...strings) :- This method is used to combine two or more string and written a new string

Eg :- let sub1 = "HTML"

let sub2 = "CSS"

let sub3 = "JS"

clg(sub1.concat(" ", sub2, " ", sub3)) // HTML CSJS

6) replace(oldString, newString) and replaceAll(oldString, newString)

Replace method is used to replace only the first occurence of the given string with a new string whereas replaceAll method replaces all the occurences of the given string with a newstring

Eg :- let str6 = "I love you only you"

clg(str6.replace("you", "myself")) // I love myself only you

clg(str6.replaceAll("you", myself")) // I love myself only myself

7) split(seperator) :- This method is used to divide the string into substring based on the seperator and return the output in the form of Array

Eg:- let str7 = "NAGACHAITANYA ♡ SAMANTHA"

clg(str7.split("♡")) // ["NAGACHAITANYA", "SAMANTHA"]

clg(str7.split("H")) // ["NAGAC", "AITANYA♡SAMANT", "A"]

8) includes(searchString, startIndex) :-

→ This method checks if the character or the string is present in the given string or not

→ If the character is present it returns true else it returns false

→ It also takes in the index value from where it should start searching for the given character or string.

→ Negative startIndex are converted to zero

Eg :-

let str8 = "Market"

clg(str8.includes("a")) //true

clg(str8.includes("x")) //false

clg(str8.includes("r", 0)) //true → starting from 0, so it is true

clg(str8.includes("r", 3)) //false

clg(str8.includes("e", -100)) //true

↳ if negative value given also it become 0 so it is true

9.) slice(startIndex, endIndex) :- This method is used to extract the portion of the string and return a new string based on start and end index values.

→ EndIndex is not included or mandatory
→ Negative index values are considered.
→ If the startIndex is greater than endIndex then it returns empty string

Eg:- let str9 = "JAVASCRIPT"
0 1 2 3 4 5 6 7 8 9
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1

```
clg(str9.slice(2)) // VASCRIPT
clg(str9.slice(3,8)) // ASCRI
clg(str9.slice(-6,-3)) // SCR
clg(str9.slice(4,-1)) // SCRIP
clg(str9.slice(8,2)) // ""
```

10.) substring(startIndex, endIndex) :-

→ This method is used to extract the portion of the string and return a new string based on start and end index values.
→ endIndex is not included or mandatory
→ Negative index values are converted to zero
→ If the startIndex is greater than endIndex then the index values will be swapped.

Eg:- let str10 = "JAVASCRIPT"

```
clg(str10.substring(2)) // VASCRIPT
clg(str10.substring(3,8)) // ASCRI
clg(str10.substring(-6,-3)) // ""
clg(str10.substring(4,-1)) // JAVA
clg(str10.substring(8,2)) // VASCRI
```

19) Arrays

→ Arrays are used to store both homogeneous and heterogeneous type of data.

→ Homogeneous refers to similar type of data

→ Heterogeneous refers to different type of data

→ Arrays are denoted by the symbol [ ] (square braces)

Creation of Arrays :

There are 2 ways of creating an Array in JavaScript

1) Literal Way
2) Constructor Object

1) Literal Way :

It involves the creation of an array using the symbol square braces [ ]

Eg :-
```
let frontendSub = ["HTML", "CSS", "JS", "React"]
let details = ["Pavan", 20, false]
clg (frontendSub) // ["HTML", "CSS", "JS", "React"]
clg (type of frontendSub) // object
```

2) Constructor Object :

→ It involves the creation of an array using the new keyword followed by array constructor which accepts the size of the array/array length

→ Array length is not fixed

Eg :-

```
let recentMovies = new Array(4)
recentMovies[0] = "Hit 3"
recentMovies[1] = "#single"
recentMovies[2] = "Court"
recentMovies[3] = "Shubham"
ricentMovies[4] = "Chaava"
clg(recentMovies) // ['Hit 3', '#single', 'court', 'Shubham', 'Chaava']
```

Accessing the Elements of an Array

→ To access the individual elements of an array we the concept of indexing

→ Index values always starts with 0 (zero)

Eg :- let chickenItems = ["65", "Biryani", "Butter Masala", "Mandi"]

```
clg(chickenItems[2]) // Butter Masala
clg(chickenItems[3]) // Mandi
clg(chickenItems[0]) // 65
```

Note :- If the index value is greater than the existing index (or) if the index value is negative it returns undefined

```
clg(chickenItems[4]) // undefined
clg(chickenItems[-1]) // undefined
```

Length :

Length property is used to find the total no. of elements present in an array

Eg : clg(chickenItems.length) //4

Multidimension / Nested Array

It is the way of writting one array inside another array

Eg : let trainer = [["Monty", "Python", 27],
["Pavan", "Core Java", 25],
["Yasin", "Sql", 28]]

clg(trainer[1][2]) //25

clg(trainer[2][1]) //Sql

clg(trainer[0][1]) //Python

Adding or Deleting the elements at starting or Ending of the Array :

To perform the above operation there are 4 methods in JavaScript

1) push()
2) pop()
3) unshift()
4) shift()

Eg :-

```
let snacks = ["Samosa", "Egg Puff"]
clg(snacks) //["Samosa", "Egg Puff"]
```

1) push()

This method is used to add the elements at the ending of the array

Eg:-

```
snacks.push("Chips", "Punugulu")
clg(snacks) //["Samosa", "Egg Puff", "Chips", "Punugulu"]
```

2) pop()

This method is used to delete the elements at the end of an array.

Eg

```
snacks.pop()
clg(snacks) //["Samosa", "Egg Puff", "Chips"]
```

3) unshift()

This method is used to add the elements at the starting of the array.

Eg :-

```
snacks.unshift("PaniPuri", "FrenchFries")
clg(snacks) //["PaniPuri", "FrenchFries", "Samosa",
                "Egg Puff", "Chips"]
```

4) shift()

This method is used to delete the element at the starting of an array.

Eg :-

```
snacks.shift()
clg(snacks) //["FrenchFries", "Samosa", "Egg Puff",
                "Chips"]
```

	Start	End
Add	unshift()	push()
Delelete	shift()	pop()

Methods

1) toString() :- This method is used to convert array to string

Eg:- let arr1 = ["Marker", "Charger", "Remote", "Laptop"]

clg (arr1.toString()) //Marker, Charger, Remote, Laptop

clg (typeof arr1) //string

2) concat(val.arry) :- This method is used to combine one or more array and return a new array

Eg:- ~~let arr2=~~

let frontend = ["Html", "Css", "Js", "React"]

let backend = ["Java", "Python"]

let database = ["Sql"]

let fullstack = [ ]

clg (fullstack.concat(frontend, backend, database))

//['Html', 'Css', 'Js', 'React', 'Java', 'Python', 'Sql']

3) join(seperator) :- It is used to combine the array elements based on seperator and return in the form of string

Eg:- let arr3 = ["RCB", "SRH", "DC", "PBKS", "GT"]

clg(arr3.join("☺")) //RCB☺SRH☺DC☺PBKS☺GT

clg(arr3.join(" ")) //RCB SRH DC PBKS GT

clg(typeof arr3.join()) // string

4) flat(depthValue) :- This method is used to convert multidimension or Nested array into single dimension or one dimension array based on the depth Value.

Eg :-

let arr4 = [1,2,3,[4,5,[6,7,[8,[9,[10,[11]]]]]]]

clg(arr4.flat()) // [1, 2, 3, 4, 5, Array(3)]

clg(arr4.flat(2)) // [1,2,3,4,5,6,7,Array(2)]

clg(arr4.flat(Infinity)) // [1,2,3,4,5,6,7,8,9,10,11]

5) at(index) :- This method returns the array ~~metho~~ elements for the specified index

→ If the index value is greater than the existing index it returns undefined

→ ~~If the~~ It will also accept the negative inde value

Eg :- let arr5 = ["65", "Biryani", "Butter Masala", "Mandi"]

clg(arr5.at(3)) // Mandi | clg(arr5.at(4)) // undefined

clg(arr5.at(1)) // Birgani | clg(arr5.at(-2)) // Butter Masala

6) slice(startIndex, endIndex) :- This method is used to extract the portion of the ~~string~~ array and return a new array based on start and index values

→ EndIndex is not included or mandatory.
→ Negative index values are considered.
→ If the startIndex is greater than endIndex it returns empty array.
→ It does not effect the original array.

Eg:-

```
let arr6 = ["Mansion House", "Magic Moments", "OakSmith",
                                "Blenders Pride", "Teachers"]

clg(arr6.slice(2)) //['OakSmith', 'Blenders Pride', 'Teachers']

clg(arr6.slice(1,4)) //['Magic Moments', 'OakSmith', 'Blenders Pride']

clg(arr6.slice(-3,-1)) //['OakSmith', 'Blenders Pride']

clg(arr6.slice(0,-2)) //['Mansion House', 'Magic Moments', 'OakSmith']

clg(arr6.slice(4,0)) //[]
```

7) includes(searchElement, startIndex) :- This method will check if the element is present in the original array or not

→ If the element is present it returns true, else it returns false
→ It also takes in the index value from where it should start searching for the character or array.
→ Negative start index values are considered.

Eg:-

```
let arr7=["Chicken Pakoda", "Mango Pickle", "Omlet", "Fish Fry",
                                                "Crispy Corn"]

clg(arr7.includes("Omlet")) //true
clg(arr7.includes("Saya Sticks") //false
clg(arr7.includes("Mango Pickle", 1) //true
clg(arr7.includes("Mango Pickle", 2) //false
clg(arr7.includes("Fish Fry", -4) //true
```

8) reverse() :- This method converts the given array into reverse order

Eg:-

```
let arr8=["Curd Rice", "Biryani", "Dal Rice", "Pulav"]
clg(arr8.reverse()) //['Pulav', 'Dal Rice', 'Biryani', 'Curd Rice']
```

9) sort() :- This method sorts the given array in ascending order based on left first digit of a numerical value

Eg:-

```
let arr9=[5, 2, 7, 1, 4, 3, 10, 30, 15]
clg(arr9.sort()) // [1, 10, 15, 2, 3, 30, 4, 5, 7]
```

* 10) splice(startIndex, deleteCount, elementToBeAdded) :-

→ This method is used to add or delete the elements from an array

→ This method effects the original array.

Eg :- Example :-

let arr10 = ["Icecream", "Gulab Jamun", "DoubleKaMeeta"]

Eg 1 :-

arr10.splice(1, 1, "kaju katli")

clg(arr10) // ['Icecream', 'kajukatli', 'DoublekaMeeta']

Eg 2 :-

arr10.splice(2, 0, "Samosa", "PaniPuri", "kachori")

clg(arr10) //['Icecream', 'Gulab Jamun', 'Samosa', 'PaniPuri', 'kachori', 'DoublekaMeeta']

Eg 3 :-

arr10.splice(-2, 2, "Apricot Delight")

clg(arr10) //['Icecream', 'Apricot Delight']

Eg 4 :-

arr10.splice(100, 0, "kunafa")

clg(arr10) //['Icecream', 'GulabJamun', 'DoublekaMeeta', 'kunafa']

✱ Array Higher Order Methods

1) forEach(fn(element, index, array)) :-

→ It is a higher order method that is used to iterate upon the array elements and perform the specific operation.

→ forEach method takes in another function (callback fn) as its argument which it returns takes 3 parameters i.e., element, index, array.

Eg :- 1

```
let arr1 = ["Monty", "Pavan", "Asnan"]
arr1.forEach((element, index, array) => {
    clg(element + " " + "Sir") // Monty Sir  Pavan Sir  Asnan Sir
    clg(index) // 0 + 2 → Monty Sir
    clg(array) // 0 ['Monty', 'Pavan', 'Asnan']
})
```

Pavan Sir → 1 | Asnan Sir → 2

Eg :- 2

```
let arr2 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
arr2.forEach((ele, ind, arr) => {
    clg(`2 * ${ele} = ${2*ele}`)
    clg(ind)
    clg(arr)
})
```

2) map(fn(element, index, array)) :-

→ It is a higher order method that is used to iterate upon the array elements and perform the specific operation.

→ map method takes in another function (callback fn) as its argument which it returns takes 3 parameters i.e., element, index, array

Example 1

```
let prices = [429, 99, 125, 200]
prices.map(
  (ele, ind, arr) => {
    clg (ele * 0.1) //42.900000000000006
                    9.9
                    12.5
                    20
  }
)
```

Example - 2

```
let mulOf2 = [1, 2, 3]
let mapOutput = mulOf2.map ((ele, ind, arr) => {
    return (`2 * ${ele} = ${2 * ele}`)
  }
)
clg (mapOutput) //['2 * 1 = 2', '2 * 2 = 4', '2 * 3 = 6']
```

Note :- The difference between forEach method and map method is that forEach method returns undefined whereas map method return an array

3) filter (fn(element, index, array)) :-

→ This method is used to filter out the array elements based on a condition and returns only the values that satisfy the condition inside an array

Eg :- let arr3 = [50, 70, 60, 95, 40]

```
let filterOutput = arr3.filter((element, index, array) =>
{
  return element > 50
})
clg (filterOutput) // [70, 60, 95]
```

4) some (fn(element, index, array)) :-

→ It itterates over the array elements and checks if atleast one of the element satisfy the condition or not

→ If the condition is satisfied it returns true else it returns false.

Eg :- let arr4 = [1, 7, 3, 0, 2]

```
let someOutput = arr4.some((element) => {
  return element > 3
})
clg (someOutput) // true
```

5) every (fn(element, index, array)) :-

→ It iterates over the array elements and checks if all of the elements satisfy the condition or not.

→ If the condition is satisfied it returns true or else it returns false.

Eg:- let arr5 = [1, 7, 3, 0, 2]

```
let everyOutput = arr5.every
((element) => {
    return element > 3
})
clg (everyOutput) // false
```

6) reduce (fn(accumulator, element, index, array), initialValue):-

→ This method is used to itterate upon the array elements and reduces to single value using a reducer function (callback function).

→ reducer method takes in two arguments that is reducer function (CB fn) and initialValue (optional, stored in the accumulator)

→ The reducer function (callback fn) takes in 4 parameters that is accumulator, element, index, array

Eg:- let reducedOutput = arr6.reduce((sum, ele, ind, arr) => {
    return sum += ele
}, 0)
clg (reducedOutput)

20) Objects

→ Objects are used to store the data in the form of key-value pair.

→ The combination of key and value is known as a property

→ Objects are denoted by { }

Creation of the Object

There are 2 ways of creating an Object in JavaScript

1) Literal Way
2) Constructor Object

1) Literal Way :- It involves the creation of an Object using the symbol { }.

Eg:-
```
let bioData1 = {
    name: "Pavan",
    age : 25,
    isMarried: false,
    exGirlfriends: undefined,
    kids: null,
    skills: ["Singing", "Dancing", "Batsman"],
    fun: () => {
        clg ("CORE JAVA TRAINER");
    },
    place: "Chaitanyapuri",
    state: "Telangana",
};
```

2) Constructor Object :- It involves the creation of an Object using new keyword followed by Object Constructor.

Eg :-
```
let bioData2 = new Object();
bioData2.name = "Monty";
bioData2.age = 27;
bioData2.isMarried = true;
bioData2.exGirlfriends = undefined;
bioData2.kids = null;
bioData2.skills = ["Dancing", "Singing", "Acting"];
bioData2.fun = () => {
  clg ("PYTHON TRAINER");
};
bioData2.place = {
    area: "Dilsukhnagar",
    state: "Telangana",
};
```

Accessing the properties of an Object

There are 2 ways of accessing the property of an Object

1) Dot Notation
2) Box Notation

1) Dot Notation :-

Eg :- clg (bioData1.name) // Pavan

clg (bioData2.skills) // ["Dancing", "Singing", "Acting"]

bioData1.fun() // CORE JAVA TRAINER

clg (bigData2.place.state) // Telangana

2) Box Notation :-

Eg :- clg (bioData1["name"]) // Pavan

clg (bioData2["skills"]) // ["Dancing", "Singing", "Acting"]

~~clg~~ bioData1["fun"]() // CORE JAVA TRAINER

clg (bioData2["place"]["state"]) // Telangana

Methods

Eg :-

```
let simpleObj = {
  name : "Someone",
  age : 18,
  place : "Somewhere"
}
```

1) Object.keys(obj) :- It is used to extract the keys name of the object in the form of an array

Eg :- clg (Object.key(simpleObj)) // ['name', 'age', 'place']

2) Object.values (obj) :- It is used to extract the values of an object and return the output in the form of an array.

Eg:- clg (Object.values (simpleObj)) // ['someone', 18, 'somewhere']

3) Object.entries (obj) :- It is used to extract the properties of an object and return the output in the form of Nested Array.

Eg:- clg (Object.entries (simpleObj)) // [['name', 'someone'], ['age', '18'], ['place', 'somewhere']]

4) Object.fromEntries (nestedArray) :- It is used to convert the nested array to an object

Eg:- let nestedArray = [["name", "XYZ"], ["age", 20], ["place", "Hyd"]]

let str = "SMILE"
clg (Object.fromEntries (nestedArray))

5) Object.assign(target, source) :- It is used to copy the values from the source (Array, String, Object) and paste it into the target object

Eg :-
```
let movieDetails = {
    movieName : "Salaar",
    hero : "Prabhas",
    heroine : "Shruthi Hasan"
}
let target = {
    yearOfRelease : 2023
}

// OBJECT TO OBJECT :
object.assign(target, movieDetails)
clg (target) // {yearOfRelease : 2023, movieName: 'Salaar', hero: 'Prabhas', heroine: 'Shruth Hasan'}

// String TO object
let str = "Lenovo"
let strObj = { }
object.assign (strObj, str)
clg (strObj) // {0:'L', 1:'e', 2:'n', 3:'o', 4:'v', 5:'o'}

// $ Array to Object
let arr = ["Something", "Nothing", "Everything", "Anything", "thing"]
let arrObj = { }
object.assign (arrObj, arr)
clg (arrObj) // {0:'Something', 1:'Nothing', 2:'Everything', 3:'Anything', 4:'thing'}
```

6) Object.seal(obj) : This method prevents/stops from adding and deleting the properties into the object, but modifying the existing property is possible

Eg:-
```
let details1 = {
    name: "Abc",
    age : 30,
    place : "Chaitanyapuri"
}
clg (details1)
Object.seal (details.1)
// Add (Not Possible)
details1.pincod = 500036
clg (details1)
// Modify (Possible)
details1.age = 25
clg (details1)
// Delete (Not Possible)
delete details1.place
clg (details1)
```

7) Object.freeze(Obj) : This method is used to prevent/stop adding, deleting and modifying of the properties in the Object

Eg :-

```
let details2 = {
    name : "Xyz",
    age : 15,
    place : "Dilsuknagar",
}
clg(details2)
Object.freeze(details2)
//Add (Not Possible)
details2.pincode = 500036
clg(details2)
//Modify (Not Possible)
details2.age = 25
clg(details2)
//Delete (Not Possible)
delete details2.place
clg(details2)
```

21) Math Object

Math is a builtin object that provides methods to perform various mathematical operations like finding maximum & minimum value, Removing the decimal part, Generating Random numbers etc.

1) PI :- It is a ratio of the circumference of a circle to its diameter

Eg:- clg (Math.PI) // 3.141592653589793

2) max(...values) :- It is used to find the maximum value from the given set of numbers

Eg:- clg (Math.max(7,2,4,3,10)) // 10

3) min(...values) :- It is used to find the minimum values from the given set of numbers.

Eg:- clg (Math.min(7,2,4,3,10)) // 2

4) sqrt(num) :- It is used to find the square root of a given number

Eg:- clg (Math.sqrt(81)) // 9

clg (Math.sqrt(-9)) // NaN

5) cbrt(num) :- It is used to find the cube root of a given number

Eg:- clg (Math.cbrt(125)) // 5

clg (Math.cbrt(-4)) // -1.5874010519681996

6) pow (base, power) :- It is used to calculate the power of a number

Eg :- clg (Math.pow (2,3)) // 8

clg (Math.pow (5,4)) // 625

7) sign (number) :- It is used to check wheather the number is positive, negative or zero

→ If positive return 1, if negative returns -1 and with the value is 0 returns 0 and -0 value return -0

Eg :- clg (Math.sign (-10)) // -1

clg (Math.sign (0)) // 0

clg (Math.sign (12)) // 1

clg (Math.sign (-0)) // -0

8) abs (number) :- This method convert negative value to a positive value

Eg :- clg (Math.abs (-10)) // 10

clg (Math.abs (-5.25)) // 5.25

clg (Math.abs (23.2)) // 23.2

9) trunc (number) :- This method is used to remove the decimal part of a number and return the integer

Eg :- clg (Math.trunc (-6.25)) // -6

clg (Math.trunc (100.34)) // 100

10) round(number) :- This method performs rounding of operation based on a condition

>0.5 → next integer value previous integer value ← <0.5

Eg :- clg(Math.round(4.55))//5
clg(Math.round(4.12))//4
clg(Math.round(4.09))//4
clg(Math.round(4.8))//5

11) ceil(number) :- This method returns the next integer value (towards right side) for the specified numb

Eg :- clg(Math.ceil(-2.8))//-2
clg(Math.ceil(-1.3))//-1
clg(Math.ceil(-0.1))//-0
clg(Math.ceil(0.9))//1
clg(Math.ceil(1.3))//2

12) floor(number) :- This method returns the befor or previous integer value (towards left side) for the specified number

Eg :- clg(Math.floor(-2.8))//-3
clg(Math.floor(-1.3))//-2
clg(Math.floor(-0.1))//-1
clg(Math.floor(0.9))//0
clg(Math.floor(1.3))//1
clg(Math.floor(2.8))//2

13) random() :- This method is used to generate a random number between the range 0 to 1

```
console.log (Math.random())
```

→ To get the random number between the range 0 to 10

```
console.log (Math.random() * 10)
```

→ To display the only integer part

```
clg(Math.round(Math.random() *10))
clg(Math.trunc(Maath.random() *10))
clg(Math.ceil(Math.random()*10)) //1-10
clg(Math.floor(Math.random()*10)) // 0-9
```

Note :- To generate random values between a specific range use the below form

```
// Math.random()*(EndRange-StartRange)+StartRange
// 400-420
clg(Math.round(Math.random()*(420-400)+400))
```

Date Object :

→ A Date Object is javascript is a builtin feature that lets you to work with dates and times

→ It represents a specific movement in time like a calender date or a clock time

→ In simple words it is a tool for handling anything related to dates and times in your code

Eg:

```
//DateObject
let date=new Date()
clg(date) // Sat May 24 2025 18:07:08 GMT+0530(IST)

//DateMethods
clg(date.toDateString()) // Sat May 24 2025
clg(date.getDate()) //24
clg(date.getMonth()) // May 4 → index value
clg(date.getFullYear()) // 2025
clg(date.getDay()) // 6 → sat

//TimeMethods
clg(date.toTimestring()) // 18:11:33 GMT+0530 (IST)
clg(date.getHours()) //18
clg(date.getMinutes()) //11
clg(date.getSecond()) //33
clg(date.getMilliseconds()) //620
```

//Set Methods

```
date.setDate(14)
date.setMonth(1)
date.setFullYear(2024)
clg(date)//Wed Feb 14 2024
```

22) For in For of

These are the loops that used to iterate upon string array and object based on a condition

Eg:-

```
let str = "Rain"
let arr = ["Samosa", "Pakoda", "Egg Puff", "Icecream", "Kachori"]

let obj = {
    name : "XYX",
    age : 20
}
```

forin loop

It is one of the loop when iterated upon

it returns for strings - index
it returns for arrays - index
it returns for objects - keys

Syntax:-

```
for(variable in str/arr/obj)
{
    //code
}
```

Eg:-

```
//String
for(let output in str){
    clg(output) //0 1 2 3
    clg(str[output]) //R a i n
}
```

```
//Array
for(let output in arr){
    clg(output) // 0 1 2 3 4
    clg(arr[output]) //"samosa", "Pakoda", "Egg Puff", "Icecream", "kachori"
}
```

```
//Objects
for(let output in obj){
    clg(output) //name age
    clg(obj[output]) // xyz 20
}
```

for of loop

It is one of the loop when iterated upon

it returns for strings - characters
it returns for arrays - elements
it returns for objects - Error

Syntax:-

```
for(variable of str/arr/obj)
{
    //code
}
```

Eg:-

```
//Strings
for(let output of str){
  clg (output) //Rain

//Arrays
for(let output of arr){
  clg(output) //"samosa", "Pakoda", "Egg Puff", "Icecream", "Kachori"

//Objects
for(let output of obj){
  clg(output) //Uncaught TypeError : obj is not iterable
}
```

23) Spread Operator & Rest Parameter

These both are the features in ES-6 version of JavaScript that involves the use of (...)

Spread Operator

It is mainly used to copy the values from one or more arrays or objects into another array/object

Eg:- //Spread Operator

```
//Arrays
let frontent = ["Html", "Css", "Js", "React"]
let backend = ["Java", "Python"]
```

```
let database = ["sql"]
let fullstack = [...frontend, ...backend, ...database]
clg (fullstack) // ['Html', 'Css', 'Js', 'React', 'Java', 'Python', 'sql']
```

//Objects

```
let obj1 = {
  name: "Arjun",
  girlfriend: "Preeti"
}
let obj2 = {
  dogName: "Preeti"
}
let finalObj = {...obj1, ...obj2}
clg (finalObj) // {name: 'Arjun', girlfriend: 'Preeti', dogName: 'Preeti'}
```

//String to Array

```
let str = "Something"
let arr = [...str]
clg(arr) // ['S', 'o', 'm', 'e', 't', 'h', 'i', 'n', 'g']
```

//Array to Object

```
let randomThings = ["Marker", "Laptop", "Mike"]
let obj = {...randomThings}
clg (obj) // {0: 'Marker', 1: 'Laptop', 2: 'Mike'}
```

//String to Object

```
let str = "Hello"
let obj = {...str}
clg (obj) // {0: 'H', 1: 'e', 2: 'l', 3: 'l', 4: 'o'}
```

Rest Parameters

→ It is used to gather the function arguments into an array.

→ Rest Parameter should always be specified as a last value in the parameter list.

Eg :-

```
//Rest Parameter
function something (a,b,c,...rest) {
  clg (a) // 1
  clg (b) // 2
  clg (c) // 3
  clg (rest) // [4,5,6,7,8,9]
}
something (1,2,3,4,5,6,7,8,9)
```

Note :-

(...) used in function parameter are referred to as Rest Parameter, apart from that it is referred to as a spread Operator.

24) setTimeout & setInterval

→ These are the Higher order Methods which are used to perform a specific task after the given duration of time

→ The time duration will be considered in terms of milliseconds 1s → 1000ms

→ They are used to perform asynchronous operations that is takes some time to perform the task

→ These take 2 arguments that is function and time in milliseconds

setTimeout(function, time in milliseconds)

This method will execute only once after the given duration of time

Eg:- //setTimeout(function, time)

```
setTimeout(()=>{
  clg("Dad, wake up")
},1000)
```

clearTimeout(id)

→ It is a method that is used to cancel out the timeout function before it is been executed

→ It takes in a parameter that is an identifier of the Timeout function that needs to be cancelled

Eg:- //clearTimeout(id)

```
setTimeout(() => {
clearTimeout(killHim)
document.writeln("<h1> RAHUL IS GOOD</h1>");
}, 3000)
let killHim = setTimeout(() => {
  document.writeln("<h1> Shoot Rahul sir</h1>")
}, 5000)
```

setInterval(function, time)

This method will execute the function multiple times at fixed intervals

Eg:- //setInterval(function,time)

```
setInterval(() => {
  document.writeln("<h1> Wakeup Dad </h1>")
}, 2000)
```

clearInterval(id)

→ This method is used to cancels out the function that is set through setInterval

→ It takes in a parameter that is an identifier of the Timeout function that needs to be cancelled

Eg :- //clearInterval(id)

```
let count = 0;
let meeting = setInterval(() => {
    document.writeln("<h1> Dad_ Wakeup </h1>")
    count++;
    if (count == 5) {
      clearInterval(meeting)
      document.writeln("<h1> Enough, stop it </h1>")
    }
}, 1500)
```

25) Browser Object Model (BOM)

→ The BOM is a programming interface that provides object & methods to interact with browser windou

→ To access the properties of browser we have window object.

Window :- The top-level browser window object. It represent the entire browser window & provides methods & properties to control & interact with it.

window object methods :- To access the method of window object there is no need to specifying window. Each and everytime.

Window Object Methods

① alert() :- The alert() method in JS is used to display a dialog box with a specified message & an OK button. It commonly used to provide users with information (or) to prompt them for input in a simple & straight-forward manner.

Syntax :- alert (message)

Eg :- let bom1 = () => {
alert ("AKKADA VELLAKU CHASTAVU")
}

② confirm() :- The confirm() method in JS is used to display a diolog box with a specified message & 2 buttons OK & cancel. It commonly used to prompt user for binary choice, typically to confirm (or) cancel an action.

Sytax :- Confirm (message)

Eg :- let bom2 = () => {
confirm ("Pelli cheskuntava leda?")
}

③ Combining alert() & confirm() :-

Eg :- let bom3 = () => {
    let answer = confirm("Last time adgtunanu")
    if (answer) {
        alert("Yes")
    } else {
        alert("no")
    }
}

④ Prompt() :- The prompt() method in JS is used to display a dialog box that prompts the user for input. It typically consists of a message, an input field, & 2 buttons ok & cancel.

The user can input text into the field & then choose to submit or cancel input. The value collected from the user will be by default taken as string.

Syntax :- prompt(message)

Eg :- let bom4 = () => {
    let num1 = number(prompt("enter a number"))
    let num2 = number(prompt("enter again"))
    clg(num1 * num2)
}

⑤ Open() :- In JS, the open() method is used to open a new browser window or tab, or to navigate the current window to a new URL. It is commonly used to dynamically open new window for various purpose such as display context, having pop-up dialog, or navigate to external links.

Syntax :- Open(url, target, features)

Eg :- let toBeClosed;

```
let boms = () => {
toBeClosed = open ("https://chat.qspiders.com/", "_blank",
                   "height=400, width=300, top=250,
                    left=500")
```

⑥ close() :- In JS, close() method is used to close the current browser window (or) tab. It is typically invoked on the window object.

Syntax :- close()

Eg :- let bomb = () => {

```
    toBeClosed.close()
}
```

⑦ Inner height x Outerheight :- This property represent the innerheight of the browser windows content area in pixels. It includes the height of the viewport ~~and width and any additional~~ but excludes the height of any browser chrome (such as toolbars & scrollbars). This property is useful for determining the available space for displaying content within the viewport.

Eg :- clg ("Innerheight is ${innerHeight}")

This property represent the outerheight of the browser window in pixels. It includes the height of the viewport as well as any addition browser chrome (such as toolbar & scrollbars). This property gives the total height of the browser window include its chrome.

Eg :- clg ("outerheight is ${outerHeight}")

(8) Innerwidth & outerwidth :- Similar to innerHeight, this property represent the innerwidth of the browser window content excludes the area in pixels. It include the width of the viewport but excludes the width of any browser chrome. It's useful for determining the available space for displaying content horizontally within the viewport.

Eg :- clg ("Innerwidth is ${innerwidth}")

Similar to outerHeight, this property represents the outerwidth of the browser window in pixels. It includes the width of the viewport as well as additional browser chrome. This property gives the total width of the browser window including its chrome.

Eg :- clg ("Outerwidth is ${outerwidth}")

Web Storage System

SessionStorage & localStorage are 2 mechanisms provided by modern web browser to store key-value pairs locally with the user's browser.

* sessionStorage :- The data stored in sessionStorage is scoped to the current browser tab/window session. It persists as long as the tab/window is open. When the tab/window is closed, the data is deleted.

Methods

① setItem() :-

Eg :- sessionStorage.setItem("name", "Pavan")
sessionStorage.setItem("age", 25)
sessionStorage.setItem("place", "Mysore")

② getItem() :-

Eg :- clg(sessionStorage.getItem("age"))
clg(sessionstorage.getItem("place"))

③ removeItem() :-

Eg :- sessionStorage.removeItem("place")

④ clear() :-

Eg :- sessionStorage.clear()

* localStorage :- The data stored in localstorage persists even after the browser is closed & reopened. It remains available until explicitly cleared by the user (or) the web applications that stored the data.

Methods

① setItem() :-

Eg :- localStorage.setItem("name", "Monty")
localStorage.setItem("age", 27)
localStorage.setItem("place", "Hyd")

② getItem() :-

Eg :- clg(localStorage.getItem("age"))
clg(localStorage.getItem("Hyd"))

③ removeItem() :-

Eg :- localStorage.clear()

History Object

Represent the browser's history stack, allowing you to navigate backward & forward through

Methods

① forward() :- goes forward by one tab

Eg :- let bom7 = () => {
history.forward()
}

② back() :- goes backward by one tab

Eg :- let bom8 = () => {
history.back()
}

~~② removeItem()~~ :-

③ go() :- method of the history object, which allows you to manipulate b/w multiple tabs of browsers

go() method forward by 4 tabs.

Eg :- let boma = () => {

history.go(4)

}

④ go() :- backward by 2 tabs

Eg :- let bom 10 = () => {

history.go(-2)

}

⑤ length :- Tells the total numbers of tabs opened in a browser

Eg :- clg (history.length)

Location Object

The location object provides a variety of properties and methods that allow you to interact with the browser's address bar, including getting the current URL, setting a new URL, reloading the page.

① href :- Fetches the entire url of a website

Eg :- clg (location.href);

② Reload() :- Reloads a page

Eg:- let bom11 = ()=>{
    location.reload()
}

③ Assign() :- Current url will assigned to new url but has the chance to go back to previous page

Eg:- let bom12 = ()=>{
    location.assign("https://www.cricbuzz.com/");
}

④ Replace() :- Current url will be replaced with new url but cannot come back to the previous page.

Eg:- let bom13 = ()=>{
    location.replace("https://www.w3schools.com/")

Navigator Object

To know the latitude and longitude values

Eg:- navigator.geolocation.getCurrentPosition((position)=>{
    clg(position.coords.latitude)
    clg(position.coords.longitude)
})

20) Document Object Model (DOM)

→ When our browser renders our html page it creates a tree like structure internally knows "DOM".

→ DOM is a standard for how to get, change, add, or delete HTML elements.

Eg :- console.dir(document)

Accessing the elements :-

To access the html elements DOM provides 2 ways

1) Direct Access
2) Indirect Access

1) Direct Access

```
clg(document.all)
clg(document.head)
clg(".title)
clg(".body)
clg(".images)
clg(".links)
clg(".forms)
```

2) Indirect Access

It provides 6 different inbuilt methods to access the html elements

1) getElementById()
2) getElementByClassName()
3) getElementByTagName()
4) getElementByName()
5) querySelector()
6) querySelectorAll()

1) getElementById() :-

getElementById is a method in JavaScript that allows you to access on HTML element in a document by its unique id attributes.

Eg :- let h1 = document.getElementById("heading")
console.dir(h1)
h1.style.color = "white"
h1.style.backgroundColor = "seagreen"

let button = document.getElementById("btn")
console.dir(button)
button.style.color = "white"
button.style.background = "royalblue"

2) getElementsByClassName :-

getElementsByClassName is another method in JavaScript that allows you to select elements in the DOM based on their class names. This method returns a live HTMLCollection of all elements in the document that have the specified class name.

Eg :- let images = document.getElementByClassName("poster")
clg(images)
for(let i=0; i<images.length; i++)
{ images[i].style.border = "10px solid red"
images[i].style.borderRadius = "50%"
}

3.) getElementsByTagName() :-

getElementsByTagName is a method in JavaScript that allows you to select elements in the DOM based on their tag name. This method returns a live HTMLCollection of all elements in the document that have the specified tag name

Eg :- let links = document.getElementsByTagName("a")

```
clg(links)
for(let i=0; i< links.length; i++)
{
    links[i].style.color="green"
    links[i].style.fontSize ="35px"
}
```

4.) getElementsByName() :-

getElementsByName() is a method in JavaScript that allows you to select elements in the DOM based on their name attributes. This method returns a live NodeList of all elements in the documentation that have the specified name attribute.

Eg :- let fields =document.getElementsByName("inputs")

```
clg(fields)
for(let i=0; i<fields.length; i++)
{
    fields[i].style.padding = "10px 30px"
    fields[i].style.backgroundColor = "maroon"
}
```

5) querySelector() :-

querySelector() is a method in JavaScript that allows you to select the first element within the document that matches a specified CSS selector.

ID :- let heading1 = document.querySelector("#heading")
clg(heading1)

class :- let poster1 = document.querySelector(".poster")
clg(poster1)

Tagname :- let link1 = document.queryselector("a")
clg(link1)

6) querySelectorAll() :-

querySelectorAll() is a method in JavaScript that allows you to select all elements within the document that match a specified CSS selector. This method returns a static NodeList containing references to all matching elements

ID :- let heading2 = document.querySelectorAll("#heading")
clg(heading2)

CLASS :- let poster2 = document.querySelectorAll(".poster")
clg(poster2)

TagName :-
let link2 = document.querySelectorAll("a")
clg(link2)

Difference between querySelector() and querySelectorAll():-

querySelector() is used when you need to select a single element or the first element that matches a css selector, while querySelectorAll() is used when you need to select multiple elements that match the css selector.

writeln() and write():

write() and writeln() methods are used to write content to the HTML document. However, they differ in how they handle line breaks

write() :- The write() method writes the specified content to the document without appending a newline character at the end.

Eg:- document.write("Good Morning")
document.write("<h1>Write Method</h1>")

→ Difference :- document.write("Hello")
document.write("Stupid Fellows")

writeln() :- The writeln() method writes the specified content to the document appending a newline character at the end.

Eg:- document.writeln("Good Mornining")
document.writeln("<h1>Writeln Method</h1>")

→ Difference :- document.writeln("Hello")
document.writeln("Stupid Fellows")

Accessing the Text Content :-

HTML

```
<p id="para">
  Lorem ipsum dolor sit amet
  <b style="display: none">
  consectetor</b> adipsicing
  elit. </p>
```

JS

```
let para = doc.getElementById(para)
```

InnerText :-
Can access only content which is visible on the ui.
Eg :- clg(`inner Text is: ${para.innerText}`).

TextContent :- can access all the content if it is hidden.
Eg :- clg(`textContent is : ${para.textContent}`)

InnerHTML :- Can access all the html structure along with the text Content inside it.
Eg :- clg(`innerHTML is : ${para.innerHTML}`)

Setting the text Content :-

HTML :-

```
<section id="container1"></section>
<script src="./dom.js"></script>
```

JS :-

```
let section = document.querySelector("#container1")
section.innerText = "Hello from innerText"
section.textContent = "Hello from textContent"
section.innerHTML = '<h1> Hello from innerHTML </h1>
<p>This is a sample para</p>
<button> Click </button>
```

Getting the attributes :-

getAttribute() and getAttributeNode() are methods used to retrieve attribute values of an HTML element, but they serve slightly different purposes.

```
let h2 = document.querySelector("h2")
```

1) **getAttribute()** :- the getAttribute() method is used to retrieve the value of a specific attribute of an HTML element.

```
Eg:- clg(h2.getAttribute("id")) // something
     clg(h2.getAttribute("class") // anything
     clg(h2.getAttribute("title") // nothing
```

2) **getAttributeNode()**

The getAttributeNode() method is used to retrieve the attribute node itself, rather than just its value.

```
Eg:- clg(h2.getAttributeNode("id")) // id="something"
     clg(h2.getAttributeNode("class")) // class="anything"
     clg(h2.getAttributeNode("title")) // title="nothing"
```

note :-
getAttribute() & getAttributeNode() are methods used to retrieves attribute values of an HTML element, but they serves slightly different purpose.

Eg:- let h2 = document.querySelector('h2')

Setting the attribute

① setAttribute() :- In JS, setAttribute is a method that allows you to set an attribute on a specified element in HTML DOM. It is primarily used to dynamic add (or) modify attribute of HTML element.

Eg:- let para = document.querySelector("p")
para.setAttribute("id", "para")
para.setAttribute("class", "paragraph")
para.setAttribute("style", "color: red; bgc: black;")
clg(para)

Remove the attribute

① removeAttribute() :- In JS, removeAttribute is a method used to remove a specified attribute from an HTML element in the DOM. This method is commonly used to dynamically manipulate the attributes of HTML elements.

Eg:- let h4 = document.querySelector("h4")
h4.remove.Attribute("id")
h4.remove.Attribute('class')
clg(h4)

Attribute & classname

let h6 = document.querySelector('h6')

① attributes :- In JS, the attribute property of an HTML element provides access to a collection of all attributes of the element. This collection is represented as a NamedNodeMap, which to an array but contains nodes with key-value pairs representing attri...

Eg:- clg(h1.attributes)

fetches the name of the class of the target element

Eg:- clg(h1.className)

Classlist

In JS, the classlist property provides an interface to manipulate the classes of an HTML element. It offers a convenient way to add, remove, toggle & check for the presence of CSS classes on an element.

Eg:- let para = document.getElementById("para")

let dealWithClass = () => {

add(class1, class2, ...);- adds one or more classes to the element if the class is already present, it will not add it again

Eg:- para.classList.add("dosa", "idli", "puri", "pohe");

remove(class1, class2, ...);- remove one or more classes from the element.

Eg:- para.classList.remove("puri");

toggle(class);- Toggles the presence of a class. If the class is present, it removes it; if it's absent, it add it.

Eg:- para.classList.~~contains~~ toggle("upma");

contains(class);- checks if the element has specific class. Returns true if the class is present false otherwise.

Eg:- clg(para.classList.contains("dosa")); // true

clg(para.classList.contains("puri")); // false

clg(para.classList.contains("upma")); // false

item(index);- Returns the classname at the specified index. This is useful when you want to access a class by it's position.

Eg:- clg(para.classList.item(2)); // pohe

clg(para.classList.item(0)); // dosa

clg(para.classList.item(3)); // upma // null

length :- returns the number of classes on the element

Eg:- alert(para.classList.length); // 3

createElement()

used to dynamically created HTML elements.

Eg:- let h1 = document.createElement("h1")
let btn = document.createElement("button")

createTextNode()

used to create a new text node. Text node are used to represent textual content within HTML elements.

Eg:- let h1 = document.createTextNode("This is heading")
let btn = document.createTextNode("This is button")

create Comment()

used to create a comment node in the HTML DOM. comment node are used to insert comments into the HTML structure, which aren't rendered by the browsers but can be useful for adding notes (or) annotations to the HTML code.

Eg:- let comment = document.createComment("This is comment");

appendChild()

appendChild is a method used to append a single node (element, text, node etc.,) as the last child of a parent node.

Eg:- h1.appendChild(h1)
btn.appendChild(btn)

append()

append is a newer method introduced in JS, & it is used to append one or more nodes or DOM string object to a parent node.

Eg:- h1.append(h1Text, btnText)
btn.append(btnText)

Displaying Content on UI

let body = document.body

body.append(h1, btn)

childNodes & children

let mother = document.body

childNodes :- contains all child nodes (including element nodes, text nodes, comment nodes etc...)

clg(mother.childNodes)

children :- contains only the element node (ie, the child element) of the current node.

clg(mother.children)

insertBefore

insertion before is a method used to insert a new node before an existing child node of a specified parent node in HTML DOM. this method provides a way to dynamically insert elements/ nodes at specific position within a parent node's children.

```
let body = document.body
let h1 = body.children[0]
let p = body.children[1]
let btn = body.children[2]
body.insertBefore(btn, h1)
body.insertBefore(p, h1)
body.insertBefore(h1, btn)
```

replaceChild(newchild, oldchild)

this method replace an existing child node (oldchild) with a new child node (newchild)

HTML

```
<ul id="hospital">
    <li> Banu </li>
    <li> mathew </li>
    <li id="tobereplace"> Ray </li>
    <li id="tobedeadsoon"> nurse </li>
</ul>
<button onclick="replaceOperation()">click to replace </button>
<button onclick="removeOperation()"> click to remove </button>
```

replaceChild(newChild, oldChild);-

```
let replaceOperation = () => {
  let hospital = document.getElementById("hospital")
  let Raj = document.getElementById("to be replaced")
  let Bantu = document.createElement("h1")
  Bantu.innerText = "Bantu"
  hospital.replaceChild(Bantu, Raj)
}
```

removeChild(child);-

The method remove a specified child node from the DOM.

```
let removeOperation = () => {
  let hospital = document.getElementById("hospital")
  let nurse = document.getElementById("tobedeadsoon")
  hospital.removeChild(nurse)
}
```

Events

Events are actions/occurrence that happen as a result of user ~~interaction~~ interaction (or) other activities in a webpage/web application. these interactions can include things like clicking on an element, however hovering over it. with the mouse, submitting a form, pressing a key on the keyboard.

addEventListener(event, function, useCapture);-

AddEventListener method is used to attach an event handler to an element. It allows you to specify a function (even listener) that should be executed when a particular event occurs on target element.

① click;- the click event is used to detect when an element (i.e, button, link (or) any clickable item) is clicked by user.

Eg:-
```
let head1 = document.getElementById("heading")
head1.addEventListener("click", () => {
  head1.style.color = "white";
  head1.style.bkg = "red";
}); head1.style.textAlign = "center";
```

② DBLCLICK :- the dblclick event in JS occurring when a user double-clicks on an element using the mouse. this event is triggered when 2 consecutive click events happen within a short time interval on the same element.

```
Eg:- let btn = document.getElementById("btn")
     btn.addEventListener("dblclick", ()=>{
        btn.style.color = "white";
        btn.style.background = "deeppink";
        btn.style.padding = "15px 20px";
     });
```

Mouseenter, Mouseleave & Mousemove

```
let area = document.querySelector("textarea");
```

a) MOUSEENTER :- the mouseevent event in JS occurs when the mouse cursor enters the boundaries of an element.

```
Eg:- area.addEventListener("mouseenter", ()=>{
        area.style.color = "white";
        area.style.background = "maroon";
     });
```

b) MOUSELEAVE :- the mouseleave event in JS occurs when the mouse cursor leave the boundaries of an element.

```
Eg:- area.addEventListener("mouseleave", ()=>{
        area.style.color = "yellow";
        area.style.background = "green";
     });
```

c) MOUSEMOVE :- the mousemove event in JS is triggered whenever the mouse pointer moves over an element

```
Eg:- area.addEventListener("mousemove", ()=>{
        let red = Math.round(Math.random()*225);
        let green = Math.round(Math.random()*225);
        let blue = Math.round(Math.random()*225);
        let finalColor = `rgb(${red},${green},${blue})`;
        area.style.background = finalColor;
     });
```

Input & Change

```
let inputfield = document.getElementById("inputfield");
```

INPUT :- the input event is fired whenever the value of an <input>, <textarea>, or <select> element changes as the user interacts with it. Unlike the change event, which fires when the element loses focus after its value has changed, the input event fires immediately when the value of element is modified.

```
Eg:- inputfield.addEventListener("input", ()=>{
    let red = Math.round(Math.random()*255);
    let green = Math.round(Math.random()*255);
    let blue = Math.round(Math.random()*255);
    let bgcol = `rgb(${red}, ${green}, ${blue})`;
});
```

CHANGE :- The input event is fired whenever the value of an <input>, <textarea>, or <selection> element changes as user interacts with it. Unlike the change event, which fires when the element loses focus after its values has changed, the input event fires immediately when the value of element is modified.

```
Eg:- inputfield.addEventListener("change", ()=>{
        inputfield.style.background = "white";
    });
    inputfield.style.padding = "20px 30px";
```

Event Propagation

It is the process of propagation (Traversing) of events from one element to another element.

Types :- (2)

① **Event bubbling phase** :-

If the event propagates from target element to parent element then it is in bubbling phase can be achieved by setting third parameter of addeventlistener

eg:- let outer = document.getElementById("outer")
let inner = document.getElementById("inner")
outer.addEventListener("click", () => {
alert("OUTER BOX")
}, false)
inner.addEventListener("click", () => {
alert("INNER BOX")
}, false)

② Event capturing phase

=> the event propagates from parent element to target element then it is in capturing phase can be achieved by setting third parameter of addeventlistener to true.

eg:- outer.addEventListener("click", () => {
alert("OUTER BOX")
}, true)
inner.addEventListner("click", () => {
alert("INNER BOX")
}, true)

stopPropagation();-

the stopPropagation() method in JS is used to prevent an event from bubbling up the DOM hierarchy.
When an event occurs on element, it normally triggers the event handler on the element & then propagates up to it's parent element & so on, potentially triggering other event handlers along the way.
By calling e.stopPropagation(), you stop this bubbling behavior ensuring that the event only affects the current target & doesn't trigger handlers on parent elements.

eg:- outer.addEventListener("click", () => {
alert("OUTER BOX")
}, false)

```
inner.addEventListener("click", (e)=>{
  clg(e)
  alert("INNER BOX")
  e.stopPropagation()
},false)
```

Promises

Promises are object used to handle asynchronous object (those operation which take some time to perform the task).

Types (States):- 3 Phase/states

1) Pending state
2) Resolved state
3) Rejected state

To handle the resolved state promise we have then() & similar to rejected state promise we have catch().

then() & catch() are higher Order Method which takes the promise result of resolved & rejected promises respectively.

```
Eg:- let p = new Promise((resolve, reject)=>{
       let meet = true;
       if(meet){
         resolve("promise got resolved")
       }else{
         reject("promise got reject")
       }
     });

     p.then((res)=>{
       clg(res + "let go inside")
     }).catch((err)=>{
       clg(err + "sorry buddy")
     })
     clg(p);
     clg(typeof p); // object.
```

Fetch, ASYNC & AWAIT

* Fetch method in JS, is used to make network request to fetch resource (like JSON data, images or any other file) from a server.
* In simple words, it helps us to extract the data from the API.

Eg:- let data = fetch('https://fakestoreapi.com/products')

```
clog(data)
data.then((res) => {
    return res.json()
}).then((res) => {
    clg(res)
})
```

* ASYNC & AWAIT are keywords introduced in ECMAScript 2017 (ES8) that work togethers to simplify asynchronous JS code.
* They help us to handle the promises gracefully.

ASYNC :- the async keyword is used to declare an asynchronous function. An asynchronous function returns a promise implicitly, which resolves to the value returned by the async function.

AWAIT :- the await keyword can only be used inside an async function. It pauses the execution of the async function until the promise passed to await is settled.

① Eg:-

```
let fetchData1 = async () =>{
  let data = await fetch("https://fakestoreapi.com/product")
  let finalData = await data.json()
  let table = document.getElementById("container")
  finalData.forEach((product) =>{
    table.innerHTML +=`
      <tr>
        <td> ${product.id} </td>
        <td> ${product.title} </td>
        <td> ${product.category} </td>
        <td> ${product.description} </td>
        <td> <img src=${product.image} height=200
                                    width=200 > </td>
        <td> ${product.price} </td>
      </tr>
    `
  })
}
fetchData1()
```

② Eg:-

```
let fetchData2 = async () =>{
  let data = await fetch("https://dummyjson.com/recipes");
  let finalData = await data.json();
  let finalRecipes = finalData.recipes;
  let table = document.getElementById("container2");
  finalRecipes.forEach((recipe) =>{
    table.innerHTML +=`
    <tr>
      <td> ${recipe.id} </td>
      <td> ${recipe.name} </td>
      <td> ${recipe.instructions} </td>
```

```
<td> ${recipe.ingredients} </td>
<td> ${recipe.cuisine} </td>
<td> <img src = ${recipe.image} height = 200
</tr>                    width = 200 > </td>
`
});
};
fetchData2();
```

Eg:-③

```
let fetchData3 = async () => {
let data = await fetch("https://api.github.com/users");
let finalData = await data.json();
let table = document.getElementById("container3");
finalData.forEach(users) => {
  table.innerHTML += `
    <tr>
      <td> ${users.id} </td>
      <td> ${users.login} </td>
      <td> ${users.type} </td>
      <td> <img src = ${users.avatar_url} height = 200
                    width = 200 > </td>
      <td> <a href = ${users.html_url}> <button>
              profile </button> </a> </td>
    </tr>
  `
})
};
fetchData3();
```

Eg:- (4)

```
let fetchData4 = async() =>{
  let data = await fetch("https://dog.ceo/api/breeds/
                          image/random")
  let finalData = await data.json()
  let section = document.getElementById("container4")
  section.innerHTML = `
      <img src = ${finalData.message}>
  `
}
```

Form Validation

Eg:-

```
let form = document.querySelector("form");
form.addEventListener("submit", (e) =>{
  e.preventDefault();
  let nameField = document.getElementById("name");
  let nameFieldValue = document.getElementById("name").value;
  let phoneField = document.getElementById("phone");
  let phoneFieldValue = document.getElementById("phone").value;
  let passwordField = document.getElementById("password");
  let passwordFieldValue = document.getElementById
                           ("password").value;
  let cpasswordField = document.getElementById("cpassword");
  let cpasswordFieldValue = document.getElementById
                            ("cpassword").value;
```

```
/* Name field validation */
if(nameFieldValue.length <3 || nameFieldValue.length >10)
{
  alert("please enter characters b/w the range 3-10");
  nameField.style.borders = "1px solid red";
} else {
    clg(nameFieldValue);
    nameField.style.borders = "1px solid green";
}

/* phone field validation */
if(!(phoneFieldValue.startsWith("6") || phoneFieldValue.startsWith("7")
    || phoneFieldValue.startsWith("8"))) {
alert("phone numbers should start with either 6/7/8");
}
if(isNaN(phoneFieldValue) || phoneFieldValue.length !=10){
  alert("please check the phone numbers");
  phoneField.style.borders = "1px solid red";
  }else{
    clg(phoneFieldValue);
    phoneField.style.borders = "1px solid green");
}

/* password field validation */
if(passwordFieldValue !== cpasswordFieldValue) {
  alert("please check your password");
  cpasswordField.style.borders = "1px solid red";
  }else {
    clg(cpasswordFieldValue);
    cpasswordField.style.borders = "1px solid green";
  }
});
```

```
/* password visibility */

let password_eye = document.getElementById
                    ("password_eye");
let passwordField = document.getElementById("password");
let ispasswordVisible = false;
password_eye.addEventListener("click", () => {
  if(ispasswordVisible){
    passwordField.type = "password";
    ispasswordVisible = false;
  }else{
    passwordField.type = "text";
    ispasswordVisible = true;
  }
});

        /* confirm password visibility */

let cpassword_eye = document.getElementById("cpassword_eye");
let cpasswordField = document.getElementById("cpassword");
let iscpasswordVisible = false;
cpassword_eye.addEventListener("click", () => {
  if(iscpasswordVisible) {
    cpasswordField.type = "password";
    iscpasswordVisible = false;
  }else{
    cpasswordField.type = "text";
    iscpasswordVisible = true;
  }
});
```