21/4/25

Java Script Topics

1) Introduction
2) History
3) Key features
4) Java Script Engine
5) way of Adding JavaScript
6) Comments.
7) Tokens
8) Data types
9) Scope
10) Hoisting and temporal dead zone.
11) Diff 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 and date Object.
22) For in & For-Of loop
23) Spread Operator & rest parameter.
24) Set Time-out & Set Interval
25) Bom (browser object model)
26) Dom (Document Object Model).

27> Events and it's propagation.

28> promises

29> Async & await

30> Fetch Method

31> Form validation.

21/4/25

* Introduction :

→ Java Script is one of the most popular programming language in the world.

→ It is Not only used as Frontend technology but also as a Backend technology.

Frontend → React Js

Backend → Node Js, Express Js.

→ Java Script is a Scripting Language that is used to add functionalities to our webpages.

* History Of JavaScript :-

→ Earlier in 1990's A company Name Netscape wanted to built a Scripting language for their Browser called Netscape Navigator.

→ They hired a person called Brendon Eich and asked him to devolop 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 a Marketing Startegy they changed it's name to JavaScript.

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

→ To Run the JavaScript code Outside the browser a person named Ryan Dahl introduced NodeJs.

* Key features :-

emp Java Script is a Scripting Language.

emp Each and Every code written in Java Script can be read, analyze and Executed in the Browser itself.

→ Java Script is a Weekly/Loosely Typed Language.

> Even if the semicolon is missing or the datatype is not specified It doesn't provide any Error.

Java	Java Script
int a=10;	a=10 & Datatype
S.O.pln (a) ;	clg (a) semicolon

→ JavaScript is a Dynamically Typed Language.

> If we declare a variable storing a numerical value, can be later reassigned with any other type of value.

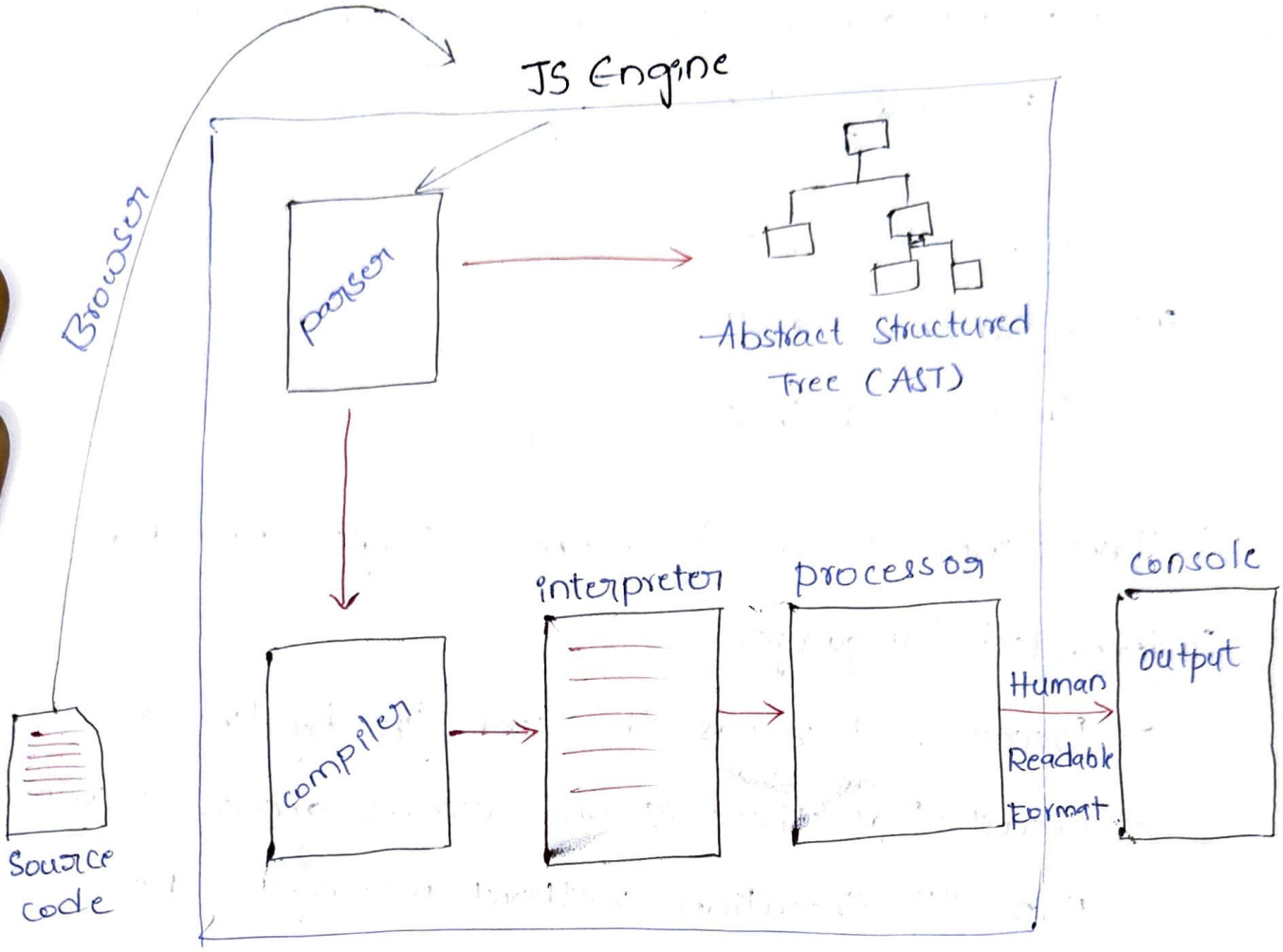
Java	JavaScript
int a = 10; ✓	a = 10 ✓
✗ a = true;	a = true ✓

→ JavaScript is a Sychronous Single Threaded / Interpreted Language.

> Because, it reads the code line by line & executes and creates. if there is any errors occurs it will stop it's Execution without moving to the next line.

22/4/25

* Java Script Engine :=



→ Browser	JS Engine
Chrome	V8
Firefox	Spider Monkey
IE → Internet explorer.	Chakra
Brave	Blink + V8

→ when the JavaScript file (source code) is executed on the browser, It is sent to the JavaScript engine.

→ In this process the source code is first sent to the parser. which stores the Original code into tree like structure known as "Abstract Structured Tree" (AST) & also divides our code it into parts (Tokens) and sents one by one to the Compiler.

→ The Compiler will check for the Syntax Errors and converts the source code into Binary codes (0's & 1's) and Sents it to the interpreter.

→ the Interpreter checks the binary code line by line and Sents it to the processor.

→ the processor will convert the binary code into Human Redable format & Prints the output inthe browsers Console.

* Way Of Adding JS :

→ There are 2 ways of adding JS, they are

1) Internal JS
2) External JS.

1) **Internal JS** : It refers to the writting of java script code inside the body tag by maintaing a script tag

Ex:

```
<body>
  <script>
    console.log("I AM INTERNAL JS");
  </script>
</body>
```

2) **External JS** : It refers to writting of the javaScript code by maintaing a seperate file saved with a Extension .js.

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

Ex:

```
<body>
<script src = "path of Js file"> </script>
</body>.
```

* Comments :-

→ comments are the piece of code i.e, not executed on the browser.

→ It is mainly used to provide (improve) the redability of the code.

Types :

There are 2 types of comments in JS,

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

23/4/25.

* Tokens :

→ Tokens are the smallest part of any programming language.

Types :

Tokens are divided into 3 categatories i.e,

1) Keywords
2) Identifiers
3) Literals.

1. **Keywords** : These are predefined (or) reserved words.

→ they should always be written in lowercase.

Ex: Console, log, if, else, const, var, let --- etc,

2. Identifiers : These are names given for the components like variables & functions.

Rules :

1) Identifier should not be same as keyword.

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

3) Except $ & _ no other spl characters should not be used.

Ex: skills, name, age1 ---etc.

3. Literals : These are the values that are stored in a variable.

Ex: 10, true, "javaScript" ---etc.

* Data Types :

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

Types : Datatypes are divided into 2 categories.

1) primitive Datatypes

1) Number
2) String
3) boolean
4) undefined
5) Null
6) BigInt
7) Symbol

2) Non-primitive D.T

1) Object.

Primitive Data types :

1. Number : It represents both integers & Decimals.

Ex
```
var Num1 = 10
var num2 = 10.25
var num3 = 10000000.56
console.log (num3) // 10000000.56
console.log (typeof num3) // number
```

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

Back ticks

Ex :
```
let str1 = 'JavaScript'
let str2 = "R"
let str3 = `10`
console.log (str2) // R
console.log (typeof str2) // String
```

3. Boolean : It represents a logical value either true/ false.

Ex
```
let bool1 = true
let bool2 = false
console.log (bool1) // true
console.log (typeof bool1) // boolean
```

4. Undefined : It refers to uninitialized variable or absence of a value for a variable.

Ex:
```
let val
console.log(val) // undefined
console.log(typeof val) // undefined
```

} diff b/w * them

5. Null : It is the intensional absence of a value for a variable

Ex:
```
let name = null
console.log(name) // null
console.log(typeof name) // object
name = "Nandu"
console.log(name) // Nandu
console.log(typeof name) // string.
```

24/3/25

6. BigInt : This datatype is used to store a large number without loosing the precision.
↓ exact value

→ To convert a number to a BigInt value specify 'n' at the end of a number.

Ex:
```
let sym = Symbol()
console.log(sym)
```
eof

```
let bigInt = 1676543219778n
console.log(bigInt)
console.log(typeof bigInt)
```

7. Symbol : This datatype is used to Generate unique id's for an object.

Ex: let sym = Symbol(1)
console.log(sym) // Symbol(1)
console.log(typeOf sym) // Symbol.

* Scopes :-

It is the Region in which the variable are accessible.

Types

They are 3 types of scopes in JS -:

1) Global Scope
2) Local / Function Scope
3) Block Scope.

1. Global Scope : Variables declared at the top level should be accessible in the same JS file.

→ Variables declared with Var keyword follows Global Scope, whereas let & const keyword follows Script Scope.

Note : Script Scope is similar to Global Scope which was introduced along with the creation of let & const keywords in ES6 version of JavaScript 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.

25/4/25

3. Block / Scope :-

→ Variables declared Inside the block should be accessible only Inside the block.

→ Variables declared with let & const keywords follows Block Scope, whereas variables declared with var keyword desn't follow Block Scope.

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

imp

* Hoisting :

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

(or)

→ It is the process of Accessing the variable even before it's 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 & const because of temporal Dead Zone (TDZ).

imp

* Temporal Dead Zone (TDZ) :

→ It is the time duration between the creation of a Variable & it's utilization.

Ex: // Variables

```
console.log(a); // undefined
var a = 10;

console.log(b) // Uncaught RefError : Cannot access 'b' before initialization.
let b = 20; // value unavailable

console.log(c)
const c = 30;
```

26/4/25

* Diff B/w var / let & Const.

	var	let	const
Scopes:	Global, Local	Script, Local, Block	Script, Local, Block.

→	Declaration without initialization is possible
	var a;
	console.log(a);
	// Undefined
		Declaration without initialization is possible
		let b;
		console.log(b);
		// Undefined.
			Declaration without initialization is Not possible.
			const c;
			console.log(c);
			// Uncaught Syntax Error:

→	ReAssigning the variables is possible
	var a = 10;
	a = "saturday"
	console.log(a) // Saturday
		→ ReAssigning the variables is possible.
		let b = 20;
		b = false;
		console.log(b); // false.
			→ ReAssigning the variables is not possible. const c = 30;
			c = "stupid";
			console.log(c); // Uncaught Syntax Error: ---

→ Redeclaration of the variables is possible.

```
var a = 10;
var a = 20;
console.log(a) // 20
```

→ Hoisting is done and value is initialized as Undefined.

```
console.log(a); // undefined
var a = 10;
```

→ Redeclaration of the variable is not possible

```
let a = 20;
let b = false;
console.log(b);
// uncaught Error:
```

→ Hoisting is done But the value is not initialized as var because of (TDZ).

```
console.log(b); //
uncaught Ref Error:
let b = 20;
```

→ Redeclaration of the variables is not possible.

```
const c = 30;
const c = "Java"
console.log(c);
// uncaught Syntax Error:
```

→ Hoisting is done But the value is not initialized because of (TDZ).

```
console.log(c); //
Uncaught Ref Error
see const c = 30;
```

* Operators :=

→ Operators are the predefined Symbols used to perform a Specific task.

→ operands are the values upon which the operation is performed.

→ The combination of operators & operands is known as "Expression".

Types :

There are 5 types of Operators in JavaScript.

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

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

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

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

5) Conditional / Ternary Operator.

28/4/25

1) Arithmetic operations.

(+ Addition)

```
console.log(2+3) // 5
console.log(typeOf (2+3)) // number
console.log (2+"3") // 23
console.log (typeOf (2+"3")) // string
console.log ("2" + '3') // 23
console.log (2+true) // 3  (2+1)
console.log (2+ false) // 2  (2+0)
console.log (2+"2"+false) // 22false
console.log (2 + 5 +"2"+ true) // 72 true
console.log (2+ 10 + 'abc') // 12 abc
```

// Substraction

```
c.log (2-3) //-1
c.log (typeof (2-3)) //numba
c.log (2-"3") //-1
c.log (typeof (2-"3")) //number
c.log ("2"-"3") //-1
c.log (2-true) //-1
c.log (2-false) //2
c.log (2-"2"-false) //0
c.log (2-5-"2"-true) //-6
console.log (2-10-'abc') // Nan - not a number
```

// Multiplication.

```
c.log (2*3) //6
c.log (typeof (2*3)) //number
c.log (2*"3") //6
c.log (typeof (2*"3")) //number
c.log ("2" * "3") //6
c.log (2* true) //2
c.log (2* false) //0
c.log (2* "2" * false) //0
c.log (2 * 5 * "2" * true) //20
clog (2*10 * 'abc') // Nan
```

// Division

```
clog (4/2) //2
c.log (typeof (4/2)) //number
c.log (4/"2") //2
c.log (typeof (4/"2")) //number
c.log ("10" /"5") //2
c.log (2/true) //2
c.log (2/false) // Infinity
c.log (2/"2"/false) // Infinity
c.log (10/5 /"1"/true) //2
c.log (2/10/'abc') // Nan
```

// Modulus

```
c.log (4%2) //0
c.log (typeof (4%2)) // number
c.log (4%"2") //0
c.log (typeof (4%"2")) // number
c.log ("10" %"5") //0
c.log (2%'abc') //NaN (Not-a-Number)
```

// Comparision (or) Relational Operators.

// (>, <, >=, <=, ==, !=, ===)

```
let p=1;
let q=2;
let r="2";

console.log (p>q) // false
c.log (p>r) // false
c.log (p<r) // true
c.log (q>=r) // true
c.log (q<=p) // false
c.log (p!=q) // true
c.log (r!=q) // false
c.log (p==r) // false
c.log (q==r) // true
c.log (q===r) // false.
```

// Logical Operators

(||, &&, !)

```
let a=1
let b=2
let c="2";
c.log (a>b || b>c) // false
c.log (a>b || b==c) // true
c.log (a>b && b>c) // false
c.log (a>b && b==c) // false
```

```
c.log (a<b && b==c) // true
c.log (!(a==b)) // true
c.log (!(b==c)) // false
```

// Assignment Operators

(=, +=, -=, *=, /=, %=)

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

c.log (a+=b) // 2010
c.log (a-=b) // 2000
c.log (a*=b) // 20000
c.log (a/=b) // 2000
c.log (a%=b) // 0
```

// Conditional / Ternary Operator

// Ex-1

```
let pellichesukuntava = true;

pellichesukuntava ? c.log ("Mi NaNNa tho
    eppudu matladadali ?") : c.log
        ("Mi amma tho eppudu
            matladuthav");
```

// Ex-2

let weekend = false;

weekend ? c.log("Thaguadam") : c.log("consider it as a weekend & Drink");

* what is NaN?

Nan → stands for Not a Number which is used to represent a invalid Mathematical Operation.

c.log (2+"2"+"false"); // Nan

* (==) :

Diff B/w == & === :-

'==' is used to compare the values whereas '===' compare both the values wrt & datatype.

Ex c.log (1 == "1") // true

c.log (1 === "1") // false.

29/4/25

* Decision Statements :

They helps to Execute the statements based on the condition.

Types :

There are 4 main types of Decision Statements in java script.

1) if
2) ifelse
3) elseif
4) switch case

1) if : Syntax :

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

Ex :

```
let area = "Ibnagar"
if (area == "chaitanyapuri")
{
    console.log ("Amma Thodu Addamga Narikestha");
}
```

2> if else: Syntax – if (condition)

```
{
// Block of code to be executed if cond is true
}
else {
// Block of code to be executed if cond is false
}
```

Example:

```
if (area == "chaitanyapuri")
{
  c.log ("Amma Todu Adanga Narikesta")
}
else {
  c.log ("Debalu padtayi")
}
```

3> elseif : Syntax : if (condition 1)

```
{
//Block of code to be executed if cond1 is true
}
else if (condition 2)
{
// Block of code tobe executed if cond1 is false & cond2 is true
}
else {
// Block of code to be executed if cond 2 is false.
}
```

Example : let amount = 50;

```
if (amount >= 350)
  {
c.log ("Pista House Biryani")
  }
else if {
   c.log ("Raghavendra Tiffins")
   }
 else if {
    c.log ("Go and eat some tiffins")
    }
  else{
    c.log ("Muskoni pg lo thinu")
   }
```

4) Switch : Syntax - switch (Expression)

```
{
case value 1: code block ;
              break;
case value 2 : code block ;
              break;
case value 3 : code block;
              break;
case value N : code block;
              break;
default :  code block;
           break;
}
```

Example-1 :

```
let trainer = "Rahul";

switch (trainer)
   {
case "Abbas" :  c.log ("Programming trainer");
                break;
case "pavan" :  c.log ("core java trainer");
                break;
case "Monty" :  c.log ("python trainer");
                break;
case "Yasin" :  c.log ("Sql trainer");
                break;
case "Shubham" : c.log ("web trainer");
                 break;
default : console.log ("Not a trainer");
}
```

Example-2 :

```
let amount = 250;

swith (true)
   {
case (amount >= 350) : c.log ("Pista House Biryani");
                       break;
case (amount > 200 && amount <=300) : c.log ("Raghavendra
                       break;
```

```
case amount >100 && amount <= 200 :
                    c.log ("Go and eat Some Tiffins");
    break;
default : c.log ("Mooskoni pg lo thinu");
}
```

30/4/25

* Loops :

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

Types : i> for loop
ii> while loop
iii> Do-while loop

i> for loop :
```
for (initialization ; condition ; updation)
{
 //code
}
```

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

ii> while loop :
```
initialization
while (condition)
{
//code
 updation
}
```

Eg:-

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

iii) do-while loop :

```
initialization
do {
  //code
  updation
  } while (condition)
```

Eg:-

```
let j=1
do {
document.writeln ("<h1> frustated---- </h1>")
  j++
  } while (j<=10)
```

Note : The difference between while loop & Do-while loop is that, while loop will only execute if the condition is satisfied whereas do-while loop atleast one time even if the condition is Not-satisfied.

Functions :

→ Functions are the block of code i.e., used to perform a specific task.

→ To Execute a function we need to call the function.

Advantages :

→ It improves the Readability of the code

→ It increases the Reusability of the code by calling it multiple times.

Types :

In General They are 2 types of functions in Java Script.

1) Function Declaration

2) Function Expression.

1) Function Declaration : 1/5/25

i) Noamal / General / common function :

Syntax :

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

function Declaration part

```
function identifier (arguments);
```

function calling statement.

Ex :
```
function anything()
{
 let a = 10;
 let b = 20;
c.log (a+b);
}
anything(); // 30
anything(); // 30
```

Behaviour of function :

1) function without parameter without return keyword

```
function greet()
 {
 c.log ("Good Evening");
 }
greet() ⟶ calling function.
```

2) function with parameter & without return keyword

```
function add (a,b)
 {
 c.log (a+b)
 }
add (10,20) // 30
add (30,40) // 70
```

3) function with parameter with return keyword

```
function mul (a,b) {
    return a*b
}
```

```
c.log(mul(2,3)) // 6
c.log (mul (10,20)) // 200
```

4> function without parameter with return keyword

```
function expressyourFeelings()
 {
    return "I Love You"
 }
c.log (expressyourFeelings()) // I Love You
```

// Extra Parameters.

```
function extraParam(a,b,c)
 {
 c.log (a+b+c) // Nan          (10+20+a)
 }
extraparam(10,20)
```

// Extra arguments

```
function extraArgs(a,b)
 {
 c.log (a+b) // 30
 }
extraArgs(10,20,30)
```

② Function Expression : It provides way to create the function without specifying the identifier (name of the Identifier).

→ Generally these type of function needs to be stored in a variable to be execute.

i) Anonymous function : These are the function that doesn't have any name for it.

syntax : variable = function (parameter)

```
{
    //code
}
variable (arguments)
```

Ex-1 :

```
let anonymous1 = function ()
{
c.log ("I am anonymous function")
}
anonymous1()
```

Ex-2 :

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

ii) 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 JS.

Syntax : Variable = (parameters) =>

```
{
// code
}
Variable (arguments)
```

Eg :-

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

Types : Based on the return, arrow function is divided into 2 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. (wantedly specifying return keyword)

Eg :-

```
let product1 = (a,b)=>
{
return a*b
}
c.log (product1 (10,10)); //100
```

2> implicit return arrow function : It says that if there is only one line statement to be executed then we can skip the return keyword as well as curly braces ({}).

Eg 1 :- let product2 = (a, b) => a*b

c.log (product2 (10,10)); //100

Eg 2 :- let add1 = (a,b) =>

c.log (a+b) //30

add1 (10,20)

* Behaviour of Arrow function :

1) Single Parameter :- whenever there is a single parameter in the Arrowfunction, we can skip the paranthesis (()) for it.

Eg :- let singleParam = msg =>

{

c.log (msg); // Hello

}

Single param ("Hello")

2) No parameter :- whenever there is no parameter in the Arrow function, we can skip the paranthesis ('()') and replace it with the underscore ('_').

3) Immediate Invoked Function Expression (IIFE):

→ these are the functions that should be executed immediately after it's declaration.

→ these functions will execute only one.

→ these functions can be normal / anonymous / Arrow but should be wrapped inside the paradthesis ('()').

Syntax : (
normal / anonymous / Arrow
)();

Eg :-
```
// Normal
(function normal (){
console.log ("Normal function")
})
();
// Anonymous
(function () {
c.log ("Anonymous function")
})
();
```

// Arrow function

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

* Higher Order Functions (HOF):

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

call Back Function :

call Back functions 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)
    }

c.log (calculator (20, 20, add)) // 40
c.log (calculator (40, 20, sub)) // 20
c.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 call back functions.

Hoisting in Functions

→ It is the process of calling or accessing the function even before it's declaration.

→ Hoisting is completely dependent on Memory Allocation but not on the output.

→ If memory allocation is done for a function then Hoisting is done else Hoisting is not done.

→ Functions like normal, Anonymous and Arrow are Hoisted But the values in Function Expression completely depend on the keywords like var, let & const.

value :

normal ⟶ entire function

anonymous, arrow > var - undefined, let, const } value unavailable.

Eg :- normal() // iam normal function

```
function normal(){
clg("I am Normal Function")
}
```

```
//* Anonymous
anonymous() // Uncaught Type Error : anonymous is not a function
var anonymous = function(){
c.log("I am Anonymous Function")
}
```

```
//* Arrow
arrow() // Uncaught TypeError : cannot access "arrow' before initializatio
let arrow = ()=>{
c.log("I am Arrow Function")
}
```

6/5/25

* Function currying :-

imp → 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 ;
 }
c.log (add1 (10,20,30));
```

→

Nested Series of function taking single argument

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

implicit return arrow function

→

```
let add3 = (a) => (b) => (c)=> a+b+c ;

c.log (add3 (10) (20) (30)) ;
```

(or)

```
let add3 = a => b => c => a+b+c
```

→ returning single value we can remove bracets.

```
c.log (add3 (10) (20) (30)) ;
```

* Execution Context :-

→ It is a area (or) the environment that holds the information about the memory Allocation for the variables & functions, regarding the scope, Execution statements etc//.

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

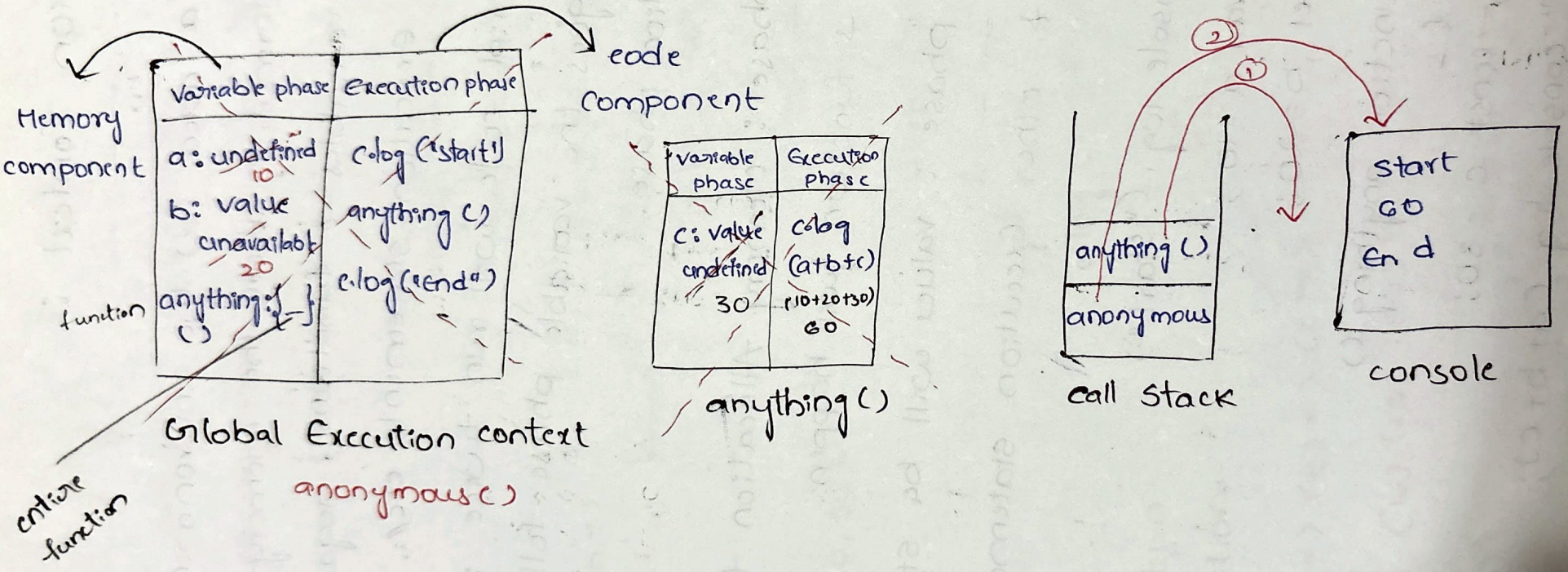
7/5/25

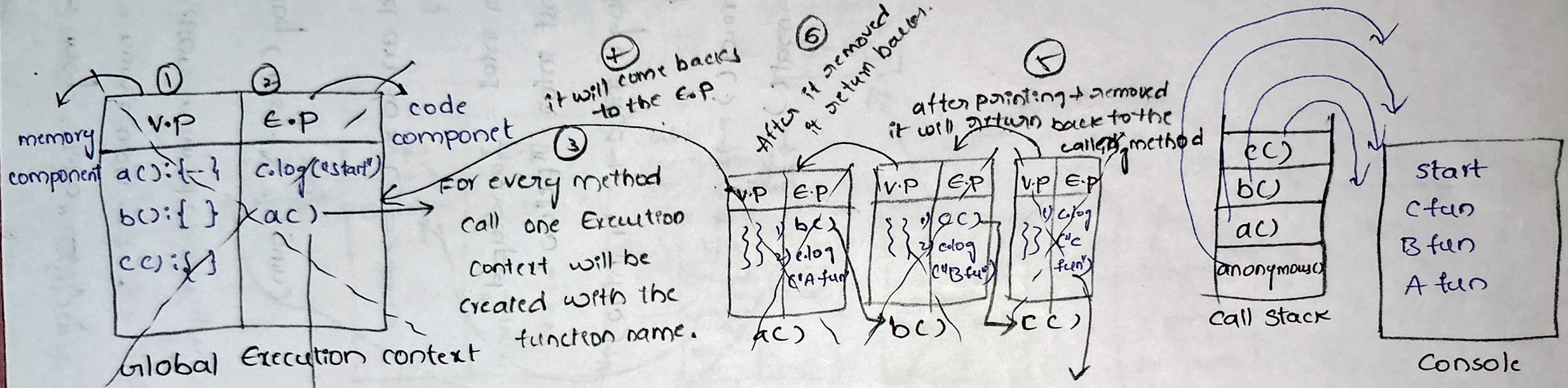
Variable phase : Memory Allocation for variables & function will happen.

Execution phase : values will be stored into the variables & other Execution statements.

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

①


Memory component
Variable phase
Execution phase
a: undefined
10
b: value unavailable
20
anything: { }
function
console.log("start")
anything()
console.log("end")
entire function
Global Execution context
anonymous()
code component
Variable phase
Execution phase
c: value undefined
30
console.log(a+b+c)
(10+20+30)
60
anything()
2
1
anything()
anonymous
Call Stack
start
60
End
console

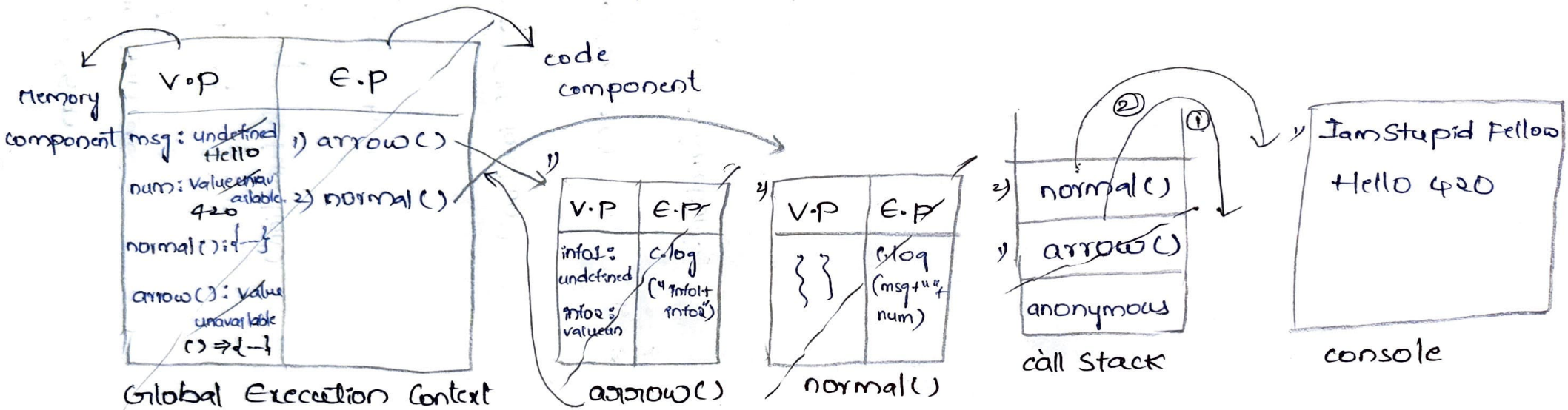


(8) so if there is next lines of code after a() it will execute the codes else it will stop the execution & removes the anonymous from the call stack.

after printing c-fun then it will be removed from the call stack (4)

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

→ click on source
→ target on first line
→ check scope & V.P & E.P
→ By clicking ↑ → this will move




```
var msg = "Hello" ⟶ 1 ✓
let num = 420 ⟶ 2 ✓
function normal() ⟶ 3 ✓
{
  c.log( msg+" "+num )
}  ⟶ 1 ✓
let arrow = () => ⟶ 4 ✓
{
  var info1 = "I am Stupid" ⟶ 1 ✓
  const info2 = "Fellow" ⟶ 2 ✓
  c.log( info1 + " " + info2 ) ⟶ 1 ✓
}
arrow() ⟶ 1 ✓
normal() ⟶ 2 ✓
```

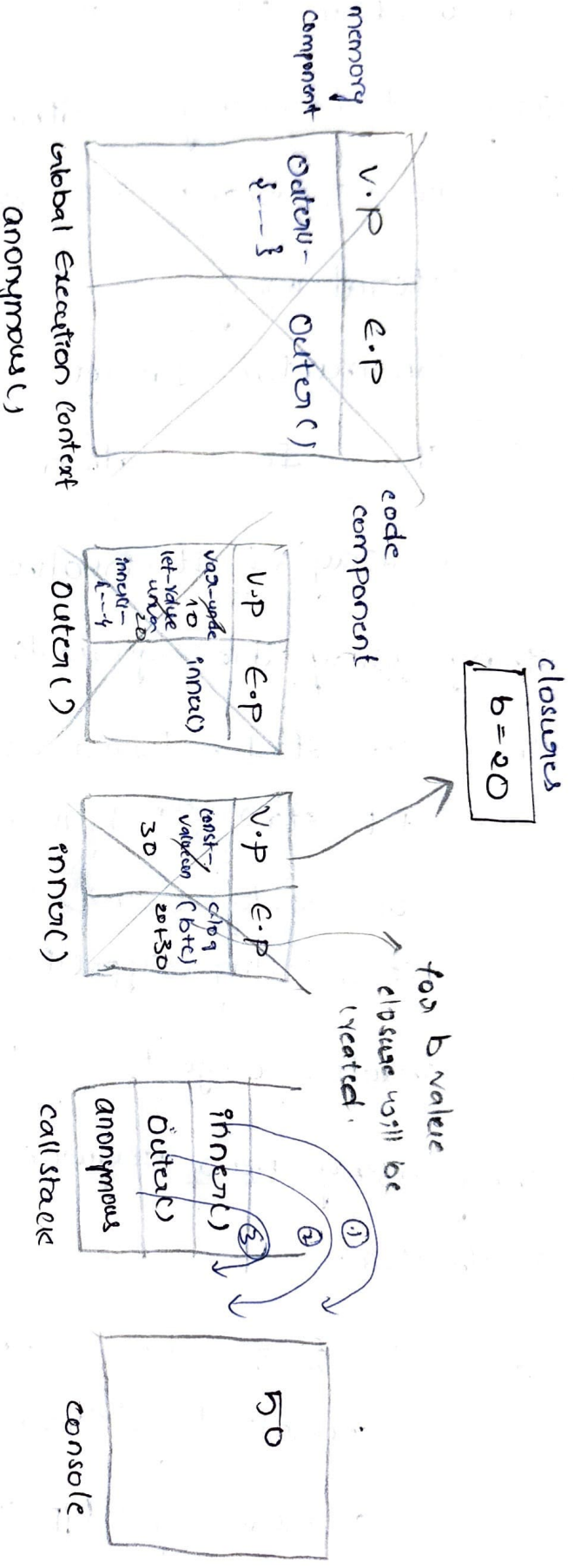
* imp

* closurs : closurs are the functions that stores the value of a outer function i.e, required by the inner function even after the completion of the execution of the outer function.

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

outer fun

inner fun



10/5/25

* Strings :

→ It is a character or Sequence of characters enclosed in '' / "" / `` (backticks).

ways of creating a String :

There are 3 ways of creating a String in JS.

1) Literal way
2) Constructor Object
3) String Interpolation (or) Template Strings.

1) **Literal way** : It involves the creation of a string using the symbols '' / "".

Ex:
```
let str1 = 'JavaScript'
let str2 = "Hyderabad".
console.log (str2) // Hyderabad
console.log (typeOf str2) // String
```

2) **Constructor Object** : It involves the creation of a String using new keyword followed by String Constructor.

Ex:
```
let str3 = new String ("Something")
console.log (str3) // String {'something'}
console.log (typeof str3) // Object.
```

→ ES6 version introduced.

3) String Interpolation / Template Strings : It involves the creation of a string using a symbol backticks which helps to dynamically render the variables inside the string using symbol $ & { }

Ex:
```
let firstName = 'Virat'
let lastName = 'Kohli'
let age = 37

c.log (`My name is ${firstName} ${lastName}
and My age is ${age}`) // My name is Virat Kohli
                          and my age is 37.
```

* Accessing the Characters Of a String :

→ To access the characters of a String we go with the concept of Indexing.

→ Index always starts with '0'.

Eg:
```
let str = "BIRYANI"
console.log (str[1]) // I
console.log (`str[5]) // N
```

Note : 1) Specifying the greater index value than the existing one. Or if the index value is -ve it returns undefined.

```
c.log (str[8]) // undefined
c.log (str[-1]) // undefined
```

2) To find the total no.of characters present in the given string we use length property:

```
c.log(str.length) // 7
```

* Built-In Methods :

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

→ Specifying the greater index value than the existing one or if the index value is -ve then it returns empty string "".

eg:
```
let str1 = "Mobile"
           012345
console.log(str1.charAt(0)) // M
c.log(str1.charAt(4)) // l
c.log(str1.charAt(6)) // ""
c.log(str1.charAt(-1)) // ""
```

2) charCodeAt(index) : This method returns the ascii value of a character present at the given index.

eg:
```
let str2 = "abc"
c.log(str2.charCodeAt(0)) // 97
c.log(str2.charCodeAt(2)) // 99
```

3) toUpperCase() & toLowerCase() : These methods converts the given characters of a string to UpperCase & LowerCase respectively.

eg: let str3 = "SoMeThiNg"

```
c.log (str3.toUpperCase()) // SOMETHING
c.log (str3.toLowerCase()) // something
```

4) repeat (count) : this method repeates the given string 'n' no.of times based on the count value.

eg: let str4 = "chai"

```
console.log (str4.repeat (3)) // chaichaichai
```

13/05/25

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

Ex:

```
let sub1 = "html"
let sub2 = "css"
let sub3 = "JS"

c.log (sub1.concat (" ", sub2, " ", sub3))
```

6) replace (old string, new string) & replaceAll (old string, new string)

- replace() method is used to replace only the first occurance of the given string with a new string where as replaceAll method replaces all the occurance of the given string with a new string.

Ex: let str6 = "I love you only you"

```
c.log (str6.replace("YOU", "Myself")) // I Love Myself only you
c.log (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 separator and return the output in the form of array.

Eg: let str7 = "nag ♡ sam"

```
c.log (str7.split("♡")) // ['nag', 'sam']
c.log (str7.split("g")) // ["na', '♡sam']
```

8) includes (Search string, start index) : 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 start index are converted to zero.

```
let str8 = "market";
c.log (str8.includes('a')) // true
c.log (str8.includes('z')) // false
c.log (str8.includes("r", 0)) // true
c.log (str8.includes("r", 3)) // false
c.log (str8.includes("e", -100)) // true.
```

9) slice (start Index, End Index) : This method is used to extract the portion of a string and return a new string based on start and end index value.

→ End index is not included.
→ Negative index values are considered.
→ if the start index > end index then it returns Empty string

Eg:-
```
let strq = "JavaScript"
c.log (strq.slice(2)) // vascript
c.log (strq.slice(3,8)) // ascri
c.log (strq.slice(-6,-3)) // scr
c.log (strq.slice(4,-1)) // scrip
c.log (strq.slice(8,2)) // ""
```

10) Substring (start index, end index): This method is used to extract the portion of a string and returns a new string based on start and end Index.

→ End index is not included.
→ Negative index values are converted to Zero.
→ If the start Index > End Index then index values will be Swaped.

```
let str10 = "JavaScript"
c.log (str10.Substring(2)) // Vascript
c.log (str10.Substring(3,8)) // Ascri
c.log (str10.Substring(-6,-3)) // ""
c.log (str10.Substring(4,-1)) // Java
c.log (str10.Substring(8,2)) // vasc
```

14/05/25

* Arrays :

→ Arrays are used to store both homogeneous as well as hetrogeneous type of data.

→ homogeneous refers to similar type of data whereas hetrogeneous refers to diff type of data.

→ Arrays are denoted by the Symbol `[ ]`.

Creation of Array :

There are two ways of creating an Array in JS

1) Literal way
2) constructor object.

1) **Literal way** : It involves the creation of an array using the Symbol `[]`.

Eg:
```
let frontendSub = ["HTML", "CSS", 'JS', 'React']
let details = ["Pavan", 20, false]

c.log (frontendSub) // ['HTML', 'CSS', 'JS', 'REACT']
c.log (typeof frontendSub) // object
c.log (details) // ["Pavan", 20, false]
c.log (type of details) // object.
```

2) **Constructor Object** : It involves the creation of an Array using **new** keyword followed by **Array** constructor which accepts the **size of an Array** / Array length.

→ Array length is not fixed.

Eg:- let recentMovies = new Array(4)

recentMovies[0] = "Hit 3"

recentMovies[1] = "#Single"

recentMovies[2] = "Shubham"

recentMovies[3] = "Court"

recentMovies[4] = "chaava"

c.log(recentMovies) // ['Hit3', '#Single', 'Shubham', 'Court', 'chaava']

Accessing Elements Of an Array :

To Access the individual elements of an Array we use the concept of indexing.

→ Index values always starts with '0'.

Eg:- let chickenItems = ["65", "Biryani", "Butter Masala", "Mandi"] → size/length = 4
(0, 1, 2, 3)

c.log(chickenItems[0]) // Butter Masala

c.log(chickenItems[3]) // Mandi

Note : If the index value is greater than the existing index or less th· if the index value is negative it returns undefined.

c.log(chickenItems[5]) // undefined

c.log(chickenItems[-1]) // undefined.

* Length : Length property is used to find the total number of elements present in an Array.

```
c.log(chickenItems.length) // 4
```

* Multi-Dimension / 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]
]
c.log(trainer[1][2]) // 25
c.log(trainer[2][1]) // Sql
c.log(trainer[0][1]) // Python.
```

15/5/25

* Adding (or) Deleting the Elements at Starting or Ending of the Array :

→ To perform the above Operation there are 4 methods in Java Script.

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

Eg:- let snacks = ["samosa", "Egg puff"]

c.log (snacks) // ['samosa', 'Egg puff']

i> push() : This method is used to add the elements at the Ending of the array.

Eg:- snacks.push ("Eggdosa", "chips", "Punugulu")

c.log (snacks) // ['samosa', 'Eggpuff', 'Eggdosa', 'chips', 'Punugulu']

ii> pop() : This method is used to delete the element at the end of an Array.

Eg:- snacks.pop()

c.log (snacks) // ['samosa', 'Eggpuff', 'Eggdosa', 'chips']

iii> Unshift() : This method is used to add the elements at the starting of an Array.

Eg:- snacks.unshift ("pani puri", "French fries", "Mirchi")

c.log (snacks) // ['Pani Puri', 'French fries', 'Mirchi', 'samosa', 'Egg puff', 'Egg dosa', 'chips']

iv> shift() : This method is used to delete the element at the starting of an Array.

Eg:- snacks.shift()

c.log (snacks)

	start	end
Add	unshift()	push()
Delete	shift()	pop()

* Methods :

1) toString() : This method is used to convert Array to String.

Eg:- let arr1 = ["Marker", "charger", "remote", "laptop"]

console.log(arr1.toString()) // Marker, charger, remote, laptop

console.log(typeof arr1.toString()) // string.

2) Concat(...arr) : this method is used to combine one or more array and return a new array.

Eg: let frontend = ["HTML", "CSS", "JS"]

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

let database = ["sql"]

let fullstack = []

c.log(fullstack.concat(frontend, backend, database))

// ['HTML', 'CSS', 'JS', 'Java', 'Python', 'sql']

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

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

c.log (arr3.join ("♡")) // RCB♡SRH♡DC♡PBKS♡GT

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

c.log (typeof arr3.join()) // string

4> flat (depth value) : This method is used to convert multi-demension / Nested Array into single ~~Arr~~ Dimension Array based on the depth value.

By default upto 1 it will take.

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

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

c.log (arr4.flat(2)) // [1,2,3,4,5,6,7, Array(0)]

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

5> at (index) : This method returns the Array elements for the specified index.

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

→ ~~If the index is~~ It will also accepts the negative index value.

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

c.log (arr5 . at (3)) // Mandi.

c.log (arr5 . at (1)) // Biryani

c.log (arr5 . at(4)) // undefined

imp *** c.log (arr5 .at(-2)) // ButterMasala.

6) Slice (start index, end index):

16/9/05

This method is used to extract the portion of an Array and returns a new array based on Start index and end index value.

→ End index is not included.

→ Negative index values are considered.

→ If the start index is greater than end index it return empty array.

→ It doesn't affect the Original array.

From left to right extract

Ex:- let arr3 = ["Mansion House", "Magic Moments", "Oaksmith", "Blender's pride", "Teachers"]

(indices: 0, 1, 2, 3, 4; negative: -5, -4, -3, -2, -1)

c.log (arr3 . slice (2)) // ['Oaksmith', 'Blenders pride', 'Teachers']

c.log (arr3. slice (1,4)) // ['Magic Moments', 'Oaksmith', 'Blenders pride']

c.log (arr3. slice (-3,-1)) // ['OakSmith', 'Blenders Pride']

c.log (arr3. slice (0,-2)) // ['Mansion House', 'Magic Moments', 'OakSmith']

c.log (arr3. slice (4,0)) // []

7) includes (search element, 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 given ~~character~~ element or ~~string~~ array.

→ Negative start index values are considered.

Ex: let arr7 = ["chicken Pakoda", "Mango Pickle", "Omlet", "Fish Fry", "Crispy Corn"]

(0, 1, 2, 3, 4 / -5, -4, -3, -2, -1)

```
c.log (arr7.includes("Omlet")) // true
c.log (arr7.includes("Soya Sticks")) // false
c.log (arr7.includes("Mango pickle", 1)) // true
c.log (arr7.includes("Mango pickle", 2)) // false
c.log (arr7.includes("Fish Fry", -4)) // true.
```

8) reverse() : This method converts the given array in reverse order.

Ex: let arr8 = ["Curd Rice", "Biryani", "Dal Kichdi", "Pulav"]

```
c.log (arr8.reverse()) // ['Pulav', 'Dal Kichdi', 'Biryani', 'Curd Rice']
```

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

Ex: let arr9 = [5, 2, 7, 1, 4, 3, 10, 30, 15]

c.log(arr9.sort()) // [1, 10, 15, 2, 3, 30, 4, 5, 7]

28/5/25

Diff B/w slice & splice

~~10~~) splice (start Index, delete Count, Elements to Be added) —

→ This method is used to modify (add (or) Delete) the Original Array.

Ex: let arr10 = ["putharekhulu", "kajukatli"]

Ex-1 :

arr10.splice(1, 1, "kakinadakhaja", "Jalebi"]

c.log(arr10) // ['putharekhulu', 'kakinadakhaja', 'Jalebi']

Ex-2 :

arr10.splice(0, 2, "Rasamalai")

c.log(arr10) // ["Rasamalai"]

Ex-3 :

arr10.splice(1, 0, "Laddu", "Soanpapdi")

c.log(arr10) // ['putharekhulu', 'Laddu', 'Soanpapdi', 'kaju katli']

* Array Higher Order Methods :

1) forEach() — It is the 'a' higher Order method that is used to Iterate Upon the Array Elements and perform a specific Operation.

→ foreach method takes in another function (call back fun) as its Argument which in return takes 3 parameters i.e, element, Index, Array.

Ex 1) forEach (function (element, index, Array))

Ex-1 :

```
let arr1 = ["Monty", "Pavan", "Yasin"]
arr1.forEach ((element, index, array) => {
    console.log(`${element} Sir`) // Monty Sir
                                  // Pavan Sir
                                  // Yasin Sir
    console.log(index) // 0
                       // 1
                       // 2
    console.log(array) // ["Monty", "Pavan", "Yasin"]*3
})
```

(string interpolation)

Ex-2 :

```
let arr2 = [1,2,3]
arr2.forEach ((element) => {
    console.log (`2 * ${element} = ${2*element}`)
})
```

// 2 * 1 = 2

2 * 2 = 4

2 * 3 = 6

// Note :

// !forEach() cannot return the values if returned gives undefined.

```
let forEachOutput = arr1.forEach((element, index, array)=>
  {
    return element
  })
console.log(forEachOutput) // Undefined
```

2) map() — It is the Higher Order ~~function~~ method that is used to Iterate upon the Array Elements and perform a Specific Operation.

→ map method takes in another function (call back fun) as its Argument which in return takes 3 parameters i.e., element, Index & Array.

Ex:

```
2) map(function(element, index, array))
   let arr2 = ["Monty", "pavan", "yasin"]
   arr2.map((element, ind, arr) =>
     {
     c.log(element + " ♡") // Monty ♡
     c.log(ind) // 012                pavan ♡
     c.log(arr) // ["Monty", "Pavan", "yasin"] *3   yasin ♡
   })
```

// Note :

// $map() returns the O/p in the form of array.

```
let mapOutput = arr2.map((element)=>
  {
  return element.charAt(0)
  })
console.log(mapOutput) // ['M', 'P', 'Y']
```

29/5/25

3) filter(function(element, index, array)) — It is a higher order method that is used to Iterate upon the array elements and returns the array of values that satisfies the condition.

→ filter method takes in another fun (call back fun) as it's argument which in return takes 3 parameters i.e., element, index & array.

Ex: filter(function(element, index, array))

```
let arr3 = [5, 1, 4, 2, 3]
let filteredOutput = arr3.filter((element)=>{
    return element>3
})
console.log(filteredOutput) // [5, 4]
```

→ here instead of return keyword if we use console.log statement then the o/p will be in the form of boolean type.

this method is like (or) operator OR

4> Some (function (element, index, array)) – It is a higher Order method that iterates upon the array elements and returns boolean value based on a condition.

→ If atleast one of the array element satisfy the condition it returns true else it returns false.

Ex- let arr4 = [5, 1, 4, 2, 3]

let someMethodOutput = arr4.some(celement) =>
{
return element > 2
})

c.log (someMethodOutput) // true

this method is like AND operator.

5> every (function (element, index, array)) – It is a higher Order method that is used to iterate upon the array elements and return a boolean value based on a condition.

→ If all of the array elements satisfy the condition then it returns true else it returns false.

Ex- let arr5 = [5, 1, 4, 2, 3]

let everyMethodOutput = arr5.every(celement) => {
return element > 2
})

clog (everyMethodOutput) // false

6) reduce (function (accumulator, element, index, array), initial value)
(is like an empty variable to store value/result.)

→ It is a Higher Order Method that is used to iterate upon the array elements and return a single value.

→ It takes in a function (reducer function) and initial value at the top-level.

→ The function in return takes 4 values i.e., accumulator, element, index & array.

Ex- let arr6 = [1, 2, 3, 4, 5]

let reducedOp = arr6.reduce (accumulator, element, index, array) =>
{
    0+=1
return accumulator += element

}, 0)
→ if we don't specify the initial value for accumulator the by default it considered the first element in the array as accumulator & element1 starts from ① which is present in 0th index.

c.log (reducedOp) // 15

30/5/25

* Objects :

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

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

→ Objects are denoted by the symbol '{ }'.

* Creation of a Object :

There are two ways of creating an object in Js,

1) Literal way

2) Constructor Object.

1) **Literal Way** : It involves the creation of a object using symbol '{}'.

ex :

```
let bioData1 = {
    name : "pavan",
    age : 05,
    hasGirlfriend : true,
    exGirlfriend : undefined,
    hasAttitude : null,
    skills : ["Flirting", "Singing", "Eating"],
    fun : ()=> {
        console.log("CORE JAVA TRAINER")
    },
    address : {
        place : "Chaitanyapuri",
        pincode : 500065
    }
}
c.log (bioData1)
c.log (typeOf bioData1)
```

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

Ex:
```
let bioData2 = new Object()
bioData2.name = "Monty"
bioData2.age = 27
bioData2.hasGirlfriend = true
bioData2.exGirlfriend = undefined
bioData2.hasAttitude = null
bioData2.skills = ["Dancing", "Singing", "Travelling"]
bioData2.fun = () => c.log("PYTHON TRAINER")
bioData2.address = {
    place : "Dilsukhnagar"
    pincode : 500064
}
```

* Accessing the properties of an Object :

There are 2 ways of Accessing the properties of an Object. 1> Dot Notation (.)

2> Box Notation ([])

1> (.) -
```
c.log (bioData1.name) // Pavan
c.log (bioData2.skills) // ['Dancing', 'Singing', 'Travelling']
bioData1.fun() // CORE JAVA TRINER
c.log (bioData2.address.place) // Dilsukhnagar
```

2) ([]) — c.log (bioData1 ["name"]) // pavan

c.log (bioData2 ["skills"]) // ['Dancing', 'Singing', 'Travelling']

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

c.log (bioData2 ["address"] ["place"]) // Dilsukhnagar

* Methods :

```
let obj = {
    name : "XYZ",
    age : 18,
    isMarried : false
}
```

1> Object.keys (obj) — It is used to extract the key names of an object and return the o/p in the form of Array.

Ex: console.log (Object.keys (obj)) // ['name', 'age', 'isMarried']

2> object.values (obj) — It is used to Extract the values of an object and return the o/p in the form of Array.

Ex: console.log (Object.values (obj)) // ['XYZ', 18, false]

3> Object.entries (obj) — This method is used to extract the properties of an Object and return the o/p in the form of nested Array.

Ex: console.log(Object.entries(obj)) // [['name', 'xyz'],
['age', 18'],
['isMarried', false]]

4) Object.fromEntries(nestedArray) – This method is used to convert nested array to an Object.

Ex:
```
let nestedArray = [
    ["name", "Abc"],
    ["age", 25],
    ["isMarried", true]
]

console.log(Object.fromEntries(nestedArray)) // {name: 'Abc'
                                                age: 25, isMarried: true
```
3/6/25

5) object.assign(target, source) – this method is mainly used to copy the values from string, array & object and place it into the targetted object.

Ex:
```
// obj to obj
let obj = {
    name: "Someone",
    age: 18,
};
let target1 = {};
Object.assign(target1, obj);
console.log(target1); // {name: 'Someone', age: 18}
```

// Array to Object

```
let arr = ["Html", "Css", "Js"];
let target2 = {};
Object.assign(target2, arr);
console.log(target2); // {0:'Html', 1:'Css', 2:'Js'}
```

// String to Object

```
let str = "Hello";
let target3 = {};
Object.assign(target3, str);
console.log(target3); // {0:'H', 1:'e', 2:'l', 3:'l', 4:'o'}
```

6) Object.seal() — This method prevents adding & deletion of ~~the~~ an Object but allows to modify the existing property.

Ex:

```
let simpleObj1 = {
    name: "Someone",
    age: 18,
};
Object.seal(simpleObj1)

// Add (Not possible)
simpleObj.place = "Hyd"
c.log(simpleObj1) // {name: 'Someone', age: 18}
// Modify (possible)
simpleObj1.name = "Monty"
c.log(simpleObj1) // {name: 'Monty', age: 18}
```

// Delete (Not possible)

a keyword.

delete simpleObj1.age

console.log(simpleObj1) // {name: 'Monty', age: 18}

7) Object.freeze() - This method prevents from adding, deleting & Modification of the properties of an Object.

Ex:

```
let simpleObj2 = {
        name: "SomeOne",
        age: 18,
    };
Object.freeze(simpleObj2)
// Add (Not possible)
simpleObj2.place = "Hyd"
console.log(simpleObj2) // {name: 'SomeOne', age: 18}
// Modify (Not possible)
simpleObj2.name = "Monty"
console.log(simpleObj2) // {name: 'SomeOne', age: 18}
// Delete (Not possible)
delete simpleObj2.age
console.log(simpleObj2) // {name: 'SomeOne', age: 18}
```

* Math Object :-

→ Math is an in-built Object that provides methods to perform Mathematical Operations like finding the max & min value, rounding of numbers, Generating random numbers etc".

1) PI — It is the ratio of the circumference of a circle to it's diameter.

Ex: c.log (Math.PI) // 3.141592653589793

2) max (... values) — This method finds the max number from the given range of values.

multiple values

ex: c.log (Math.max (2,5,3,1,4)) // 5

3) min (... values) — This method finds the min number from the given range of values.

ex: c.log (Math.min (2,5,1,3,4)) // 1

4) sqrt (number) — This method finds the Square root of the given number.

Square root

ex: c.log (Math.sqrt (81)) // 9

c.log (Math.sqrt (16)) // 4

5) cbrt (number) — This method finds the cube root of the given number.

cube root

ex: c.log (Math.cbrt (125)) // 5

c.log (Math.cbrt (8)) // 2

6) pow (base, power) – this method finds the power value for a given number.

Ex: c.log (Math.pow (2, 3)) // 8

c.log (Math.pow (8, 2)) // 64

7) trunc (number) – this method removes the decimal part & returns the integer part for a given number.

Ex: c.log (Math.trunc (10.04)) // 10

c.log (Math.trunc (-0.05)) // -0

8) abs (number) – this method converts the negative value to a positive value.

Ex: c.log (Math.abs (-5.05)) // 5.05

c.log (Math.abs (10)) // 10

9) sign (number) – this method checks whether the given number is +ve, -ve or zero.

→ If the nbr is +ve it returns 1, if -ve returns -1 & if it is zero it returns zero.

Note: If the value is -0 it returns -0.

Ex: c.log (Math.sign (-10.05)) // -1

c.log (Math.sign (05)) // 1

c.log (Math.sign (0)) // 0

c.log (Math.sign (-0)) // -0

10> round (number) – This method performs the rounding of Operation based on a specific condition.

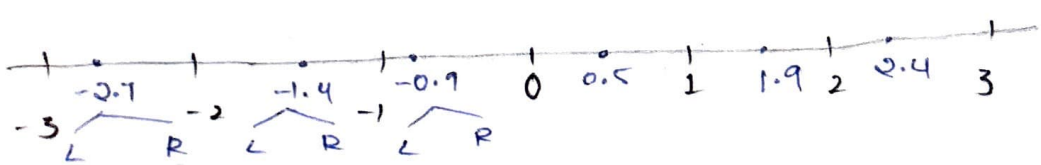
≥ 0.5 // 5
< 0.4 // 0

→ if the decimal part of a number is greater than or equal to 0.5 it returns the upcoming integer value. & if the decimal part is less than 0.5 it returns the previous integer value.

Ex: c.log (Math.round (4.56)) // 5
c.log (Math.round (4.49)) // 4

11> ceil () – This method returns the upcoming integer towards right side.

Ex: c.log (Math.ceil (0.5)) // 1
c.log (Math.ceil (1.9)) // 2
c.log (Math.ceil (2.4)) // 3
c.log (Math.ceil (-0.9)) // -0
c.log (Math.ceil (-1.4)) // -1
c.log (Math.ceil (-2.7)) // -2



12> floor () – This method returns the previous integer towards left side.

Ex: c.log (Math.floor(0.5)) // 0

c.log (Math.floor(1.9)) // 1

c.log (Math.floor(0.4)) // 0

c.log (Math.floor(-0.9)) // -1

c.log (Math.floor(-1.4)) // -2

c.log (Math.floor(-0.7)) // -3

4/6/25

mostly used to generate OTP

13) random() – This Method Generates a random value b/w the range 0 to 1.

Ex: → console.log (Math.random())

→ To Generate a random number b/w the range 0 to 10 we specify as

Ex: → console.log (Math.random()*10)

→ To Deal with the decimal part we can make use of the following methods.

ex:→ c.log (Math.round (Math.random()*10))

c.log (Math.trunc (Math.random()*10))

c.log (Math.floor (Math.random()*10))

c.log (Math.ceil (Math.random()*10))

Note : To Generate Random values B/w A Specific range use the below formula.

Math.random()* (End Range - Start Range) + Start Range

100 – 105

```
console.log (Math.random()*(105-100)+100)
console.log (Math.round (Math.random()*(105-100)+100))
```

* **Date Object** — Date is a object in Java Script, that provides the information wort date & time.

```
let date = new Date()
console.log (date)  → displays the Current date

// Date Methods
console.log (date.toDateString())
console.log (date.getDate())
console.log (date.getMonth())
console.log (date.getFullYear())
console.log (date.getDay())

// Time Methods
console.log (date.toTimeString())
console.log (date.getHours())
console.log (date.getMinutes())
console.log (date.getSeconds())
console.log (date.getMilliseconds())

// Set Methods
date.setDate (30)
date.setMonth (7, 15)
```

```
date.setFullYear(1947, 7, 15)
console.log(date).
```

* for-in & for-of Loop :

→ These are the loops that are used to iterate upon strings, arrays and object based on a condition.

```
let str = "RCB"
let arr = ["EE", "SALA", "CUP", "NAMBU"]
let obj = {
    name : "Someone",
    age : 18
}
```

* **for-in loop** – It is a loop when iterated upon

String → index

array → index

Objects → ~~in~~ key

Syntax :
```
for (variable in str/arr/obj)
{
  // codes
}
```

Ex :
```
// Strings
for ( let output in str) {
  c.log (output) //012
  c.log (str [output]) // R C B
}
```

```
// Arrays

for (let output in arr)
  {
    c.log (output) // 0 1 2 3
    c.log ( arr [output]) // 'EE' 'SALA' 'CUP' 'NAMBU'
  }

// objects
for (let output in obj)
    {
    c.log ( output) // name age
    c.log (obj [output]) // SomeOne 18
    }
```

* for-of loop – It is a loop when Iterated upon

  strings ⟶ characters

  arrays ⟶ elements

  objects ⟶ error

syntax : for (variable of str / arr / obj)

```
{
// code.....
}
```

Ex :

```
// Strings
for ( let output of str)
  {
   c.log (output) // 'R' 'C' 'B'
  }
```

```
// Arrays
for (let output of arr)
  {
  c.log (output) // 'EE'  'SALA'  'CUP'  'NAMBO'
  }

// Objects
 for (let output of obj)
    {
    c.log (output) // Uncaught Type Error : obj is not iterable.
    }
```

5/6/25

* Spread Operator & Rest Parameter :

→ These are bol features in ES6 version of JavaScript that involves the use of ... (three dots).

* Spread Operator — It is mainly used to copy the values from one or more array / Object and place it into another array / Object.

Ex:

```
// Arrays
let frontend = ["Html", "Css", "Js", "React"]
let backend = ["Java", "Python"]
let database = ["Sql"]
let fullStack = [...frontend, ...backend, ...database]
c.log (fullStack) // ['Html', 'Css', 'Js', 'React', 'Java',
                     'Python', 'Sql']
```

// Objects

```
let Obj1 = {
    name : "SomeOne",
    age : 18
    }
let obj2 = {
    place : "Somewhere"
    }
let Obj3 = {
    ...obj1
    ...obj2
    }
c.log (obj3) // { name: 'Someone', age : 18, place : 'Somewhere' }
```

// String to Object (obj.assign) method to convert str into obj

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

// String to Array

```
let arr = [... str]
c.log (arr) // fna ['H', 'E', 'L', 'L', 'O']
```

// Array to Object

```
let array = [ "Red", "Green", "Blue"]
let object = {...array}
c.log (object) // { 0 : 'Red', 1: 'Green', 2 : 'Blue'}
```

* Rest Parameter : It is used to gather the function arguments into an array.

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

Ex: function display (a,b,c,... rest)
↳ can be any name

```
{
 c.log (a) //1
 c.log (b) //2
 c.log (c) //3
 c.log (rest) // [4,5,6,7,8,9]
}
display (1,2,3,4,5,6,7,8,9)
```

6/6/25

* Set Time Out & Set Interval :-

→ These are the higher Order Methods that are used to perform asynchronous Operations..

→ Asynchronous Operations are those that take some time to perform the task.

→ These both methods takes in 2 arguments i.e., function, time duration, in milliseconds.

* Set time Out — Settime Out (fn, time)

→ This method will execute the function only once after completion of the Given time Interval/Duration.

Ex: Set Time Out (()=>{
 c.log ("Hello") → only one time it will print
},5000)

* clearTimeOut (id) — This method is used to cancel out the time out method before it is being executed.

→ here the id refers to the variable which stores the timeout.

ex: let nottoGreet = SetTimeOut (()=> {

clearTimeOut (wishHim)

console.log ("Not to Wish Him")

}, 3000)

let wishHim = SetTimeOut (()=> {

c.log ("Good Morning")

}, 5000)

2> Set Interval (fn, time) — This method executes the function multiple times after the completion of Given time Interval.

ex: Set Interval (()=> {

document.writeln ("<h1> Bye </h1>")

}, 1000)

* clearInterval (id) — This method cancels out the function created through Set Interval.

→ here the id refers to the variable which stores the Interval.

Ex :

```
let count = 0
let askForBike = setInterval (() => {
document.writeln ("<h1> Gift Me A Bike </h1>")
count ++
if (count == 5)
{
clearInterval (askForBike)
document.writeln ("<mark> Mooskon; First
}        Engineering Complete Chey </mark>")
}, 2000)
```

6/6/25

* Browser Object model (BOM) :-

→ The Browser Object Model (BOM) is a programming interface that provides objects and methods to interact with browser window.

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

* Window — The top-level browser window object.

→ It represents the entire browser window and provides methods and properties to control and interact with it.

```
console.log (window)
```

* alert() — The alert() method in JavaScript is used to display a dialog box with a specified message and an OK button.

→ It is commonly used to provide users with information or to prompt them for input in a simple and straightforward manner.

→ The syntax for the alert() method is as follows:

Syntax: alert(message)

Ex: let bom1 = ()=>{
    alert("ALA VELAKU CHASTAVU")
}

* Confirm() — The confirm() method in Js is used to display a dialog box with a specified message and two buttons: OK & Cancel.

→ It is commonly used to prompt users for a binary choice, typically to confirm or cancel an action.

→ The syntax for the confirm() method is as follows:

Syntax: confirm(message)

Ex: let bom2 = ()=>{
    confirm("Do You Really Love Me")
}

* Combining alert() & confirm()

```
let bom3 = ()=> {
let output = confirm("Do You Really Love Me")
                    ↳ boolean value
if (output)
  {
alert ("Thank You For Confirming")
 }
else {
 alert ("Just Get Lost")
 }
}
```

* prompt() — The prompt() method in JavaScript is used to display a dialog box that prompts the user for input.

→ It typically consists of a message, an input field, and two buttons : OK & Cancel.

→ The user can input text into the field and then choose to submit or cancel the input.

→ The value collected from the user will be by default taken as String.

Syntax : prompt (message)

Ex :
```
let bom4 = ()=> {
    let num1 = Number( prompt ("Enter a number"))
    let num2 = Number (prompt("Enter another number"),
} console.log (num1 + num2)
```