Java Script

* Why we ule Should JS

→ If we want to add dynamic functionality in our application we have to use Java Script.

- In Sepetember 1995, Brendan Eich a programmer at Netscape developed a programming language in just 10 days

- This language was initially called Mocha, then live Script and later Java Script

- In December 1995 Netscape and Sun (the organization that owned Java) had a licenle agreement regarding JS (then live Script).
  Hence it was renamed to JS and specifically "Java" Script because Java was the most popular language at that time and this would help the marketing of Java Script as well

- JS grew rapidly since then and in 1997 ECMA was given the responsibity to create a specification for the language

- ECMA : European Compute manufacturer association.

- They have introd ES 6 in 2015

→ Defination

1] Dynamic in nature :- JS is dynamically typed which means that variable are not bounded to specific data type ; you can assign different type of value to a variable at different point in your code

ex:- var a = 100
a = " Hello "

2] Synchronous :- Synchronous mean sequence, where the code is executed line by line

3] Single threaded :- It mean it can perform only one task at a time because JS is having only one CallStack.

4] Interpreted :- It will check or read the code line by line from top to bottom, right to left

5] weekly or loosely typed :-

Because semicolon (;) are not mandatory

ex:- consolo.log (a)

Note :- Sometimes it is mandatory to use semicolon (;)

ex:- Console.log (a); Consolo.log(a)

classmate

Date
Page

* ES 6 Features current version is ES 23

→ Keyword :- Let & Const
→ Function :- Arrow
→ operator :- {a, b} & [a, b]
→ object :- Map & Set
→ Classes
→ Promises
→ Symbol
→ Default parameter
→ String.includes()

→ Array.from()
→ Array.key()

* Define Java Script:

JavaScript is a object-based and functional programming language which is

i] Dynamic in nature.
ii] Synchronous programing language
iii] Single threaded
iv] Interpreted
v] weakly or loosely typed programing language.

→ JS is a high level dynamic, interpret programing language used for enhancing the functionality of web page.

classmate
Date
Page

* variable , identifiers

Type of variable
Var / let / const

var a = 100
Variable identifier value

→ what are variable :- Variable are use to store the data or info

→ what are identifier :- Identifier is a name given to any variable, class, function, object, array etc.

→ Rules of identifier :-

i] Identifier names must start with letter, underscore (_) & dollar ($)

ii] Identifier name ~~must~~ cannot start with numbers

iii] JavaScript is a caseSensitive : So Capital letter and small letter varies.

iv] Keywords cannot be written as identifier
ex:- for , break etc

*

Variable	Declare	Initialize	Declare & Initialize in single line	R-Dec	R-Int	R-Int & R-Dec
Var	✓	✓	✓	✓	✓	✓
let	✓	✓	✓	✗	✓	✓
Const	✗	✗	✓	✗	✗	✗

→ ex: Var

Dec :- var a ✓

INT :- a = 100 ✓

conslog (a)

var a = 100 ✓

R.dec => var a ✓

R.int :- var = "Hi" ✓

R.dec & int :- var a = "Hi" ✓

→ ex: let

Dec :- let a ✓

int :- a = 100 ✓

Consolelog (a)

let a = 100 ✓

R.Dec :- let a ✗

R.int :- a = "hi" ✓

R.int & Dec : let a = "Hi" ✗

→ ex: const

Dec :- const a ✗

int :- a = 100 ✗

const a = 100 ✓

* If we want to write the script tag inside the head then use 'defer' keyword
. The defer attribute specifies that the script should be executed later.

* Linking the JavaScript to html page

. At the end of the body write the script tag with source attribute.

```
<script src="  " > </script>
<script src="  " defer > </script>
```

* output methods of JS

i] console.log ( ) } print in console
ii] Document.write ( ) } print on screen
iii] alert ( ) } pop up
iv] prompt ( ) } Take input data
v] confirm ( ) } pop up

Note :. • prompt is used to accept user data
• will accept the data in the form of string.

→ To check the type of an object
(type of ....)

* Datatype :

It is used to define which type of data we are going to use.

in JavaScript basically they have 2 type
i] primitive / immutable / object-dereferencing
ii] Non-primitive / immutable / object referencing

Datatype

Primitive / immutable / obj (basically)

i] Number
ii] Null
iii] Boolean
iv] Bigint
v] Symbol
vi] String
vii] undefined

ex:-

```
var a = "Hello"
Console.log (a)
a = "cute"
Console.log (a)
```

Not:- It is going to create an another object mean now it is having 2 object with same name
a = "Hello"
a = "cute"

non-primitive / mutable / obj

i] Array
ii] Object
iii] function

ex:-

```
var array = ['a', 'b']
Console.log (array)
array[0] = "c"
Console.log (array)
```

:- Here it will replace the one character ~~the~~ in the same array

* Type of Conversion / Casting

i] implicit
ii] Explicit

i] implicit :- When internally conversion is happening is called as implicit type - casting.

```
clg (2+2)          4
clg (2-2)          0
clg (2*2)          4
clg (2/2)          1
clg (2+"2")        22
clg (2-"2")        0
clg (2-"h")        NaN
clg (true-false+1) 2
clg (true*true)    1
clg (false*true)   0
clg ("2"-"2")      0
clg ("hi"*2)       NaN
clg (false/true)   0
clg (2++"2")       4
clg ("2"+2-2)      20
clg (2+'2'-2)      20
clg (true-"1.5")   -0.5
clg ("hi"**2)      NaN
clg ("2"-"4")      -2
clg (typeof 2)     Number
clg (typeof 2+2)   Number
clg (typeof 2-2)   Number NaN
clg (typeof true+true)
        Boolean true
clg (typeof false-null) NaN
clg (true-f) => 1

true = 1
false = 0
```

```
clg (type of 2+2) Number2
clg (type of Null) object
clg (type of Number) function
  "      "    "  String) "
  "      "    "  BigInt) "
  "      "    "  Symbol) "
                undefined) undefined

let number1 = 273456789 // number
    number1 = 273456789n // Bigint

clg (type of Number()) number
            String()) string
            Boolean()) Boolean
            boolean()) undefined
            window()) object
            true) Boolean
            undefined) "
            true+1) Boolean 1
```

:: Explicit :

If externally we are going to convert one datatype to another datatype

ex: let num = Number (Prompt ("enter number"))

or

let num = ParseInt (prompt ("enter num"))

* Template literal or Interpolation

It is used to print the variable data.

Template literal are string literals in JS that allow embedded expression and multi-line strings.
They are enclosed by (`) backticks

* Hoisting

Before declaration and initialization if we are trying to print the identifier, if the identifier is declare by using var keyword is called Hoisting and we will get undefined

ex:
```
clg (a)      //undefined
var a = 100
clg (a)      //100
```

• -> TDZ (Temporary Dead Zone)
we will get uncaught Reference error in let and const

ex:
```
let (a) clg (a)      //Reference
var     let a = 100
        clg (a)      //100
```

* Scope :-

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

i] Global Scope

When we are going to declare any identifier by using var keyword the identif will present inside global scope.

ex:-
```
var a = 100
let b = 100
{
    console.log (a)
    console.log (b)
}
```

ii] Script Scope

When we are going to declare any identifier ~~with~~ by using let and const keyword the identif will present inside script scope, any var can point or access

ex:-
```
let b = 200
console.log (b)
```

iii] Block Scope

The scope restrict the variable that is declared inside a specific block, from access by the outside block. The let and const keyword will not work below they are in block scope

v] closure
vi] module

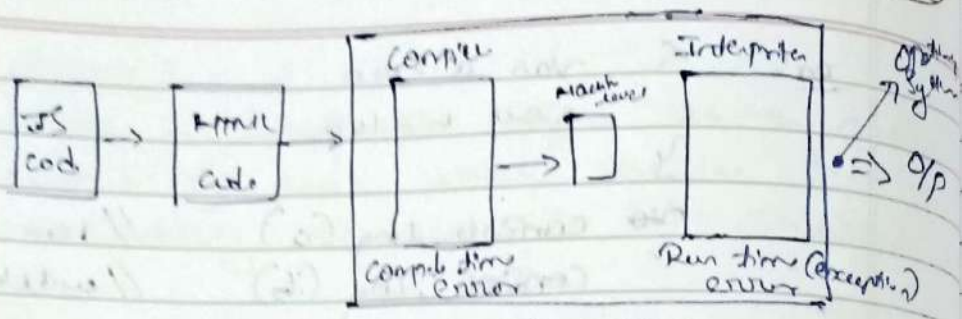
ex:
```
{
    var a = 100
    let b = 100
}
console.log(a)      // 100
console.log(b)      // undefined error
```

iv] local scope or function scope
In this scope what ever we write or any variable we cannot access out of that block.

ex:
```
function a() {
    const a = 100
    var b = 100
}
a()
console.log(a)      // undef error
```

* JavaScript engine
JS engine is used to execute the JS code. This engine help to convert our JS program into computer understandable language.

chrome - V8
Edge - chakra
Mozilla firefox - Spider Monkey
Safari - JS core webkit



Browser

Function

It is used to perform multiple operation or task

i] Anonymous function

— A function without name is called as anonymous function

ex: function () {
console.log ("hi")
}

ii] Named function

→ function with name is called as named function.

ex: function abc () {
cons.log ("hi")
}
abc ()

iii) Function as a expression

A function expression is, is a syntax construct that create a function object with an expression or another syntax construct.

When we are using return statement inside function that we must console the function calling statement.

When we are not using return statement inside function then no need to write the function calling statement.

Ex: Implicit

```
var a = function() {
        cons.log("Hello")
    }
a()                          // Hello
console.log(a())             // Hello
                             undefined
```

explicit

```
var b = function() {
        console.log("Hello")
        return "Hi"
    }
b()                          // Hello
console.log(b())             // Hello
                             Hi
```

iv] Higher order function || Callback function

- HOF :- A function which accepts another function as a argument is called as HOF

- Callback :- A function which is ~~called~~ passed as a ~~callback~~ function value to another function is called callback

HOF:	Callback
function HOF ( )	function callback ( )
{	{
console.log ("Hi")	console.log ("Helo
}	}
HOF (callback ()) → argument	

Note: Here a function is calling another function as an argument.

	or
function HOF (a) ← parameter	function callback ( )
{	{
conso.log (a())	conso.log ("Helo
conso.log ("Hi")	}
}	
HOF (callback)	

Ex:
```
function mathof (a, b, callback) {
    consolog ('Hi')
    consol.log (a)
    consol.log (b)
    callback (10, 50)
}
mathof (10, 20, function (x, y) {
    cons.log (x)
    consol. (y)
    consol.lg (x+y)
})
```

Ex:
```
let a = parseint ("enter 1st number")
let b = parseint (enter 2nd number)
function addition (a, b, callback) {
    callback (a, b)
}
addition (a, b, function (a, b) {
consol.log (' #{a} and #{b} is {a+b}')
})
```

V] ~~Arrow~~ function ( ) =>

Arrow function allow us to write shorter function syntax

Ex: let a = ( ) => {
    consol. log ("Hi")
}
a()

rules

i) Arrow function with no arguments

ii) Arrow function with 1 argument
(~~curly brace~~ parenthis are not required)

iii) Arrow function with multiple arguments
(~~curly brace~~ parenthis are required here)

iv) Arrow function can be write without curly braces
i.e, if it is have more then one, then it will shuffle or reverse the output.

v) Arrow function for single output we can write without curly braces

vi) Arrow function accept return statement only in curly braces

vii) Arrow function return is not mandatory, we can print our statement directly.

```
() => console.log("hi")
```

or

"hi"

ex:- function mat (a, b, c) {
c (a, b)
}
mat (10, 20, (a,b) => {
cons.log (a+b)//30
})

* function hoisting

function hoisting is possible only in normal function

ex: function abc () {
cons.log ("Hi")
}
abc ()

abc()
function abc () {
cons.log ("Hi")
}

vi] IIFE (Immediate invoke function expression or self invoking function)

IIFE are JavaScript functions that are executed immediately after they are defined. They are typically used to create a local scope for variable to prevent them from polluting the global scope.

ex:- (function () {
cons.log ("Hi")
}) ()

vii) Nested function

If a function is created in a function is called nested function

ex:-

```
function grandparent ( ) {
    console.log ("Hi")
const | let | var a = 20

    function parent ( ) {
        console.log ("Hello")
        cons.log log (a)
        var b = 30

        function child ( ) {
            cons log (b)
        }
        child( )
    }
    parent ( )
}
grandparent ( )
```

Note:- Here the child can access the parent property but a parent cannot access the child

parent
↓
child

classmate

Date
Page

• closure

A closure (scope) in JavaScript is a function that has access to variable in its parent scope

or

When inner (child) function is trying to access outer (parent) function variable the closure will create and closure will create only for parent function

ex:- parent (closure is created in parent)
↓
child

* Global execution context (GEC)

whenever the JS engine receives a script file, it first creates a default execution context known as the global execution context (GEC)

The GEC is the base/default execution context where all JS code that is not inside of a function gets executed.

Every JS file has only 1 GEC

* Function execution context (FEC)

whenever a function is called, the JS engine creates a different type of execution context known as FEC within the GEC to evaluate and execute the code with that function

i.e :- Every function creates a ~~di~~ own FEC.

ex: 3 function → 3FEC
10 function → 10FEC

* In GEC and FEC we are have 2 type of phase
i] Creation phase ii] execution phase

i] creation phase :- when data is stored.

ii] execution phase :- the data is executed or pointer is broken

i] G.E.C

Creation	Execution
a [x] 100	clg("start")
b [x] 200	
c [x] 300	
g() (x)	

ii] FEC (grand parent)

Creation	execution
this window	clg("gp")
d [x] 400	clg (a)
e [x] 500	clg (b)
f [x] 600	clg (c)

iii] parent (F.E.C)

Creation	execution
this window	clg ("parent")
g [x] 700	clg (d)
h [x] 800	clg (e)
i [x] 900	clg (f)

iv] child (FEC)

Creation	execution
	clg("child")
	clg (g)
	clg (h)
	clg (i)

Output:
Start
grand parent
100
200
300
parent
400
500
600
child
700
800
900

* operators : operators are used to perform operation

i] Arithematic operation
→ As are used to perform arithmetic on number.

operator	Description	Example
+	Addition	10+20 ⇒ 30
−	Subtraction	10−20 ⇒ 10
*	Multiplication	10*20 ⇒ 200
**	exponention	5 * 2 ⇒ 25 (5x5
/	Division	10 / 2 ⇒ 5
%	Modulus	21 % 5 ⇒ 1
++	Increment	10 ++ ⇒ 11
--	decrement	10 -- ⇒ 9

ii] Assignment operator :-
→ As assign value to Javascript variable.

operator	Example	Same as
=	X = Y	X = Y
+=	X += Y	X = X + Y
−=	X −= Y	X = X − Y
*=	X *= Y	X = X * Y
/=	X /= Y	X = X / Y
%=	X %= Y	X = X % Y
**=	X **= Y	X = X ** Y

ex. i] let a=10,
console.log (a)
var b = a
console.log (b)

ii] let x=10; let y=10
console.log (x + y)
x = x + y
x += y
clg (x)

iii] let x=10 ; let y=10
clg (x - y)
x = x - y
x -= y
clg (x)

iv] let x=10 ; let y=10
clg (x * y)
x = x * y
x *= y
clg (x)

v] let x=10 ; let y=10
clg (x / y)
x = x / y
x /= y
clg (x)

vi] let x=10 ; let y=10
clg (x % y)
x = x % y
x %= y
clg (x)

vii] let x=10 ; let y=10
clg (x ** y)
x = x ** y
x ** = y
clg (x)

ii] comparision operator :-
Co are used to compare the values

operator	Description
= =	equal to
= = =	equal value and equal type
! =	not equal
! = =	not equal value or not equal type
>	grater than
<	less than
> =	greater than or equal to

<= ?	less than or equal to ternary operator

ex: i] let a = 10, let b = "10"
cong (a == b)
true

ii] let a = 10 let b = "10"
clg (a === b)
false.

iii] let a = 10 let b = "10"
clg (a != b)
false

iv] let a = 10 let b = "10"
clg (a !== b)
true

v] let a = "20" let b = 10
clg (a > b)
true

vi] let a = 10 let b = 5
clg (b < a) //t
clg (a < b) //f

vii] let a = 10 let b = 10
clg (a >= b)
true
clg (a > b) //f

viii] let a = 10 let b = 20
clg (a <= b) //f
clg (b <= a) //f

iv] String operator

ex: i] let a = "H", let b = "H"
cg (a == b)
True

ii] let a = "h"
let b = "H"
cg (a > b) //T
∴ ASCII values variee
cg (b < a) //T

iii] let a = "Hello"
let b = "world"
cg (a + " " + b)

// Hello world

v] logical operator

operator	Description
&&	
||	
!	

Table &&

T T = T
T F = F
F T = F
F F = F

i] let a = 10
let b = 20
cg (a > 5 && b > 20)
false

Table ||

T T = T
T F = T
F T = T
F F = F

ii] let a = 10
let b = 20
cg (a > 5 || b > 20)
//T

T = 1
F = 0

Looping Statement

i] For
ii] while
iii] do - while

i] For loop

To check or run the statement to a certain period with increment is called as for loop

ex:-
```
for (i=0 ; i<=5 ; i++)
{
    clg (i)
}
```

ii] While loop

Ex:-
```
let i = 0
while (i<=5)
{
    clg (i)
    i++
}
```

iii] Do while

If we want to run or print atleast once

```
let i = 0
do
{
    clg (i)
    i++
} while (i<=5)
```

* Conditional Statement

i] If
ii] If else
iii] else

i] If

It will the check the Statement if it is true then it will print or else it will not print the Statement

```
let a = 20
if ( a > 19 ) :
{
    console.log ("a is greater tha 19")
}
```

ii] else

If the condition is checked wthe, if it gets fale tha its will executed the else part.

ex:- check the age for driving license

```
Var age = 18
    if (age < 19) {
        cons. log ("you age is correct")
    }
    else {
        cons. log ("you are kid")
    }
```

iii) else if

```
let age = 20
if (age <= 12) {
    clg ("you are kid")
}
else if (age < 25) {
    clg ("you can visit")
}
else {
    cons.log ("Invalid")
}
```

* Break Statement / Switch Case

In break statement they are multiple cases where if any case gets matched then it will give output else invalid. Only one statement is passed

```
let day = new Date().getDay()

Switch (day) {
    Case 0:
        clg ("Sunday")
        break;

    Case 6:
        clg ("Saturday")
        break;

    default :
        clg ("Invalid")
        break;
}
```

* Strings

A String is a set of collection of character

→ Types to write String

i] let a = 'Hello'
ii] b = "Hello"
iii] c = `Hello`
iv] d = String ("Hello")
v] e = newString ("Hello")

ex:- let e = new String ("Hello")
clg (e.valueOf())

* String Methods

i] String.length // This is read all property

let a = "Hello"
clg (a.length) //5 (o)

ii] at (Position)
clg (a.at(3)) // l (undef.)

iii] charAt (position) empty

clg

iv] charCodeAt (position) // ASCII value

var a = "a"

clg (a.charCodeAt [0]) // 97 (NAN)

v] slice (from, TO) // (execute)

var a = "Hello" (empty)

clg (a.slice (0, 2)) // He

clg (a.slice (0)) // ello

clg (a.slice (-1, -3)) //

vi] Substring (start, end) empty

The difference is that start and end values less than 0 are treated as 0 in Substring()

If start is greater than end, it will swap (4, 1) to (1, 4)

clg (a.substring (0)) // Hello

clg (a.substring (-5, 3)) //

clg (a.substring (2, -1) //

vii] substr (start, length) empty

negative length in substr() method

This code demonstrate an attempt to use a negative length parameter in the substr() method, which is invalid. It expects a positive length value

clg (a.substr (5, 5))

viii]. toupper case

let a = " prasad "
clg (a. touppercase ())

ix] tolowercase

let a = " PRASAD "
clg (a. tolowercase())

x] concat

let a = "Hello"
let b = "World"
clg (a. concat (b))
clg (a + b)

xi] trim ()
Remove white space from an string from starting and ending

let a = " Hello "
clg (a. trim ()) // space are removed

xii] trim start ()

Remove white space from the starting of the string.

xiii] trim end ()
Remove white space from the string of the ending

classmate
Date
Page

xiii] Padstart( )
padstart method pads a string from the start.

let a = "5"
let result = a.padstart (5, "x")
clg (result) // xxxx5

xiv] Padend ( )
Padend method pad a string from the end

let a = "5"
let result = a.padend (5, "x")
clg (result) // 5xxxx

xv] Repeat ( )
Repeat the string with a no of copies

let a = "hello"
clg (a.repeat (5)) // HelloHelloHelloHelloHello

. This does not effect to original string

xvi] Replace
Replace a particular string, it require 2 inputs old string and new string

Let a = "Hello world"
clg (a.replace ("world", "window"))

classmate
Date
Page

xvii Replace All ~~sapty~~

It will replace all the search value with the replace value

let a = "Hello world Hello world"
clg (a.replaceAll ("world", "Window"))

xviii /i flag "it is used to (insensitive)
let a = "HELLO"
clg (a. replace (/hello /i, "window"))

xix /g flag "global

wherever the same name are there it will be changed "use for multiple"

To use both i and g.

clg (a. replace (/Hello /ig, "Hi"))

xx index of ( ) (-1)

It will return (-1) if the text is not found.

let a = "HELLO i am prakash"
clg (a.indexOf ("prakash"))

25] Split (" ") -> String to array
26] Join ( ) -> array to string
27] fromcharcode ( )

classmate
Date
Page

xxi] lastIndexOf ( ) (-1)

let a = " hi i am pralad and pralad "
clg (a.lastindexof (" pralad ")

If the word is not found then return -1.

xxii] Search ( ) (-1)
The Search method Search a String from a String (or a regular expression) and return the position of the match.

let a = " Hello I am pralad "
clg (a. Search (" pralad ") // return index

xxiii] match ( ) Null
The match method returns an array containing the results of matching a String against a String (or a regular expression)

let a = " Hello I am pralad "
clg (a. match (" pralad "))

24] include ( ) (T || F)
It returns true if a String is present or else false.
It does not accept regular expression like (i/g)
ex: let a = " Hello I am pralad "
clg (a. include (" pralad ") // true

Array.

Array is a huge block of memory which can store both homogenous and heterogenous type of data.

or

It is a collection of homogenous and heterogenous type of data.

```
let array = [10, 20, 30]      //homogenous
console.log (array)

let array = [10, "Hello", True]   //heterogen
console.log (array)
```

i] Array literal

```
let array = [10, 20, 30]
consolelog (array)
```

ii] Array Constructor way

```
let a = new Array (10)
console.log (a)      // (10) [empty x10]
```

Note! - Whenever we pass a number in the first with only one element then it will take al length so try to "pass 2 or more"

```
let a = new Array (10, 10)
console.log (a)      // (2) [10, 10]
```

iii] Array.of()

let a = Array.of(10)
clg (a) // [10]

Methods

Array helper method | Array high order Method

i] PUSH

In this a new element is inset at the end of the array, where we can add multiple element in the array

ex: let a = [10, 20, 30]
clg (a.push(40))
clg (a) // 10, 20, 30, 40

it will effect the original array

ii] Pop [undefined]

Remove an last element from an array and return it. If the array is empty, undefined is returned and the array is not modified.

let a = [10, 20, 30]
clg (a.pop()) // 10, 20

Note: multiple element cannot be removed
∴ It returns the removed element.

iii) Unshift

Insert new elements at the start of an array, and return new length.

let a = [10, 20, 30]
console.log (a.unshift (100, 200)) // [100, 200, 10, 20, 30]

∴ we can pass single or multiple elements.

iv) Shift [undefined]

Removes the first element from an array and returns it. If the array is empty, undefined is returned and the array is not modified.

let a = [10, 20, 30]
console.log (a.shift ()) // [20, 30]

∴ If array is empty then it is undefined

v) Slice

It does not effect the original array

let a = [10, 20, 30, 40]
let b = ~~console~~ a.slice (1, 3)

console.log (b) = // [20, 30]
console.log (a) = // [10, 20, 30, 40, 50]

5) flat map

classmate
Date
Page

vi] Splice
Removes element from an array and, if necessary, inserts new elements in their place, returning the deleted elements

The Splice () method adds / or removes array elements.
The Splice () method overwrites the original array

let array = [10, 20, 30, 40, 50]
array.Splice (1, 2, "Hi")
index no
how many elements you want to remove
want to add new

Console.log (array).

7] Reverse
It will reverse the array elements and it will effect the original array

let array = [10, 20, 30]
clg (array.reverse ())

8] flat ()
It is used to convert multi dimension array into single dimension array.

let array = [[[[[[[[10], hi]]]]]]]

let b = array.flat (Infinity)
Console.log (b)
for no

* Array higher order method

i] find ()

(undefined)

- It returns the value of the first element that passes the test
- It execute a function for each array elm
- It returns undefined if no elements are found
- It does not execute the function for empty elements
- It does not change the original array

Note :- It will return the value if that condition passes

```
let array = [10, 20, 30, 40]
let result = array.find ((m) => {
        return m > 20
    })
```

output // 30

```
clg (result)
```

∴ If the element is passed through the condition rest elements will not be executed it will be terminated.

classmate

Date
Page

ii] find ~~array~~ Index ( ) (-1)

- It executes a function for each array element
- It returns the index (position) of the first element that passes the condition or test
- It returns -1 if no match
- It does not execute the function for empty array
- It does not change the original array

```
let array = [10, 20, 30, 40]
let result = array.findIndex((m) => {
        return m > 20;
})
clg (result) // output => 2
```

iii] Some ( ) (T/F)

- It check if any array element pass the test (provided as a callback function)
- It executes the callback function once for each array element
- It returns (true) if condition pass else (false)
- It will not execute for empty array
- It does not change original array

```
let array = [10, 20, 30]
let result = array.some((m) => {
        return m > 20
})
consolog (result)      output // true
```

iv] every method ( ) (T/F)

- It executes a function for each array elem
- It return true if the all elements pass the condition
- It return false if the function return false for one element
- Does not execute for empty array
- Does not chang orig array.

```
let array = [100, 20, 30, 40, 50]
let result = array.every ((m) => {
        return m > 50
    })
clg (result)          // false
```

∴ Note: even a single elem fail also we get false

v] map ( )

- It create new array (or).
- It does not effect the orig array for.
- It will not execute the function for empty array.

```
let array = [10, 20, 30]
let result = array.map ((m) => {
        return m > 20
        or
        return m + 10
    })
clg (result) = [false, false, T, T, T]
             = [20, 30, 40]
```

Note: It will perform operation on each and every element of array

vi] filter ( )

- Returns the element of an array that meet the condition specified in a callback function.
- It does not effect the original array
- Does not execute for empty array

```
let array = [10, 20, 30, 40]
let result = array.filter ((n) => {
        return n > 10
    })
cg (result)        // [20, 30, 40]
```

vii] reduce ( )

- It perform task to each and every element
- It returns a single value. the function accumulated result.
- It does execute function for empty array
- It does not change original array.

```
let array = [10, 20, 30, 40]
let result = array.reduce ((acc, val) => {
        return acc + val
    })
cg (result)        // 210
```

viii] forEach

- Performs the specified action for each element in an array.
- we should not use return statement inside forEach(), if we write it give undefined
- It call a function for each elmt
- It not execute for empty arry
- It not effect the original array.

```
let array = [10, 20, 30, 40]
let result = array.forEach((m) => {
        clg (m + 10)
        or
        clg (m + "Hi")
    })
clg                    // [20, 30, 40, 50]
```

ix] Sort

- It is used to sort the data.

```
let array = [100, 200, 30, 40]
let res = array.sort((a, b) => {
        return a - b
    })
clg (res)              // 30, 40, 100, 200
```

classmate
Date
Page

x] for of

```
let array = [100, 20, 30]
for (let anyname of array) {
    clg (anyname)
}                          // 100
                           // 20
                           // 30
```

xi] for in

```
let array = [10, 20, 30]
for (let key in array) {
    clg (key)
}                          // 0
                           // 1
                           // 2
```

∴ Here first we get index position, if we want to know the value then we use entries, keys()

[in for of loop]

⇒ entries
which give index position and value

ex. (2) [0, 10]
(2) [1, 20]
(2) [2, 30]

⇒ values
which give the value of the array

ex. 10
20
30

⇒ keys
which give the only index position.

ex. 0
1
2

effect original → • push, pop, unshift, shift
splice, sort, sort

Xii] rest parameter :

The is rest parameter allow a identifier to accept an indefinite number of arguments as an array.

gathering multiple in single

```
let array = [10, 20, 30, 40, 50]
let [b, c, ...d] = array
clg(b)  // 10
clg(c)  // 20
clg(d)  // 30, 40, 50
```

∴ The spread or rest parameter are should in the last, else not work

Xiii] Spread operator :

It allow us to expand elements

The is spread operator (...) allow us to quickly copy all or part of an existing array.

```
let array = [10, 20, 30, 40]
let res = [... array]
clg(res)
```

→ findLast()

It will give the last element of array

ex:- let array = [2, 17, 27, 56, 99, 37]
let a1 = array.findLast(() => true
clg(a1) // 37

Array which not affect original array
• map • filter • reduce • concat • slice
• splice • join() • find • findindex
• some • every • Array.from

→ Set

It is used to collection of unique values. each value can only occur in a set. The value can be of any type, primitive or obj

→ Destructuring

It is feature that allows you to extract values from arrays, objects, or function parameters and assign them to variable.

→ copyWithin( )

In this method, it copies elements to another position in an array, overwriting existing values.

ex: let a = ["a", "b", "c", "d", "e", "f"]
let b = a.copyWithin(2, 0, 2)
console(b) // ["a", "b", "a", "b", "e", "f"]
∴ [target, start, end]

→ includes ( )

• This method return true if an array contains a specified value.
• It used to check if a specific string or element is present in another string or array.

ex: let a = [10, 20, 30]. let b = [30, 40, 50]
if (!b.includes(a[i])) {
array.push(a[i]) } // [10, 20]

→ Array.isArray (value or variable name)

Used to check whether it is array or not it give in boolean

Object

→ Object is a real world entity which is used to store the data in form of key and value pair.

→ How many way can create an object

1] object literal

```
let obj = {
    emp-id : 200
    emp-name : "Prafed"
}
```

Curd operations :- i] insert new key
ii] how to fetch key
iii] how to update
iv] Delete

i] insert new key
→ obj.gender = "Male"

ii] how to fetch key from an object
eg (obj.emp-name)
eg : (obj["emp-name"])

iii] update
obj.emp-name = "pralad Sarga"

iv] Delete
delete obj.gender

2] constructor

```
let obj = new Object ({emp-name: 'pralad'})
clg (obj)
clg (obj.emp_name)
```

3] object. Create

```
let obj = object.Create ({emp_id: "420",
                          emp_name: "majnu"})
clg (obj.emp_name)
```

4] object. defineProperty ( )

Note! Here we can create only 1 obj

```
let obj = Object.defineProperty ({}, "emp", {
    value : {
        emp_name : 'pralad',
        emp_location : 'Banglor',
    },
})
clg (obj)
clg (obj.emp.emp_location)
```

5] object. define properties ( )

Note! It is similar to previous one but here we can add multiple object.

```
let obj = object.defineProperties ({}, {
    emp : {
        value : {
            emp_id : "101",
            emp_name : "pralad"
        }
    },
```

property

```
emp2: {
        value: {
                emp_id : "102"
                }
        }
clg (obj)
```

6] Constructor function

```
function demo (id, name){
        this.id = id
        this.name = name
    }
let demo 1 = new demo (101, "Praloid")
let demo 2 = new demo (102, "Amit")

clg (demo1)
clg (demo2)
```

classmate

Date
Page

Object Methods

```
let obj = {
        emp-id : "420",
        emp-name : "Laila",
        emp-location : "US"
    }
```

∴ note in the value may be anything Boolean, number, String, array, function

ex: emp-skills : ["HTM", "CSS"].
emp-instaid : () => {
clg ("Hi")

i) object.keys ( )
clg (Object.keys(obj)) // ["emp-id", "emp-name"]
∴ The output will be in array format.

ii) object.values ( )
clg (object.values (obj))
// ["420", "Laila", "US"]
∴ The out put will be in the array format

iii) object.entries ( )
clg (object.entries (obj))
// ["emp-id", "420"]
["emp-name", "Laila"]
∴ The output in array format

iv] object.freeze ()

clg (object.freeze (obj))

Note :- Here after using freeze meth, we cannot overwrite or make other or it

- Canot be updated
- Cannot insert new key
- Cannot delete

∴ Cannot perform curd operation

v] Object.isfrozen()

∴ It is used to check that weather the obj is freezed or not, the output will in the form of Boolean

clg (object.isfrozen(obj))

// True or false

vi] object.seal ()

∴ It is similar to freeze where here also some rule are the

- we can update the value of key
- cannot insert
- cannot delete

clg (object.seal (obj))

viii) object.isSealed(obj)

It is used to check that weath it is sealed or not, it return in boolean

```
clg (Object.isSealed(obj))
// True or false
```

viii) object.assign()

∴ The assign method is used to copy the value and properties from one or more source objects to a target object.

```
let obj2 = Object.assign({}, obj)
clg (obj2)
// The output is form of object
```

(target, source)

∴ if we need to print in array

```
let obj2 = object.assign([], obj)
clg (obj2)
// the output in form of array.
```

target :- it is the target object to which value and properties have to be copied

Source :- It is the source object from which value and properties have to be copied

ix Destructing

let obj = {
emp-name : 'prasad',
emp-location : 'Banglore',
}

∴ Destructing
let {emp-name, emp-location} = obj

→ Rest parameter

→ we can chge name

let {emp-name, ... emp} = obj

clg (emp) // all rest data is stored in

→ Spread operator

let a = {... obj}
clg (a)

→ If we want to use spread operator in console use curly brace
clg ({... obj})

classmate

Date
Page

X] Copy of an object

i] let obj = {
emp-id : 101
emp-name : 'praad'
}
clg (obj)

i] Store in an variable
let obj2 = obj
clg (obj2)

ii] object. Constructor
let obj3 = new object (obj)
clg (obj)

iii] object. Create
let obj4 = object. Create (obj)
clg (obj 4)

iv] object. defineproperty
let obj 5 = object. definepropties ({}, 'emp', {
value : obj
})
clg (obj 5)

v] object. definepropertys
let obj 6 = object. definepropties ({}, {
emp : {
value : obj
}
})
clg (obj 6)

vi) object.assign

```
let obj7 = Object.assign({}, obj)
clg (obj7)
```

vii) Spread operator

```
let obj8 = {...obj}
clg (obj8)
```

viii) rest parameter

```
let {...obj9} = obj
clg (obj9)
```

ix) StructuredClone()

```
let obj10 = StructuredClone(obj)
clg (obj10)
```

* Shallow copy and Deep copy

```
let obj = {
        id : 101
        name : "Prasad"
        address : {
                city : "bangalore" }
        }
```

i) ~~Assign operator~~ Store in a variable.

```
let obj1 = obj
clg (obj1)
```

→ obj1.name = "Swap"

```
clg (obj1)
clg (obj)
```

∴ It will effect the both main & nested obj

classmate

Date
Page

ii] Assign operator / Shallow copy

let obj2 = Object.assign({}, obj)
clg (obj2)
obj2.name = "Suraj" // It will both obj2
obj2.address.city = "Mysore"

note:: we can chang the nested one but not the original object.

The Main object is not effect, but it effect the nested or child object

iii] Spread operator // Shallow copy

let obj3 = {... obj}
clg (obj3)

~~let~~ obj3.name = "Suraj"
obj3.address.city = "Mysore"

note! The Main object is not effected but the nested or child only get effect.

* Deep Copy

i] JSON.parse (JSON.stringify)

→ let obj5 = JSON.parse (JSON.stringify (obj))
clg (obj5)

~~obj5~~ obj5.name = "Suraj"
obj5.address.city = "Nyem"
clg (obj5)
clg (obj)

Note: It will not effect both main body and child.

ii] Structured clone
let obj5 = structuredclone (obj)
clg (obj5)

Obj.name = "Laila"
Obj.city.location = "Nyem"
clg (obj5)
clg (obj)

Note: It will not effect both main and child object.

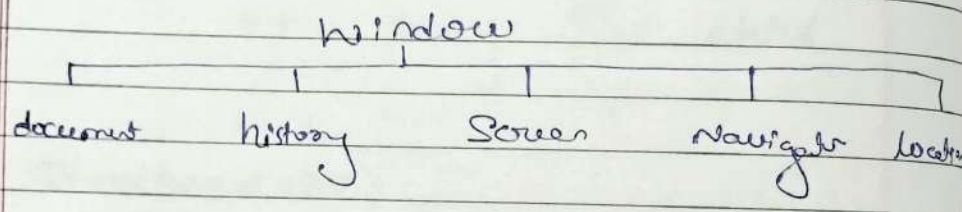
Shallow Copy	Deep Copy
i] Shallow copy stores the references of object to the original memory address	It stores copies of the object value
ii] It reflects change made to the new object in the original object	It doesn't reflect change made to the new object in the original object.
iii] It is faster	It is slower

* BOM

→ BOM :- Browser object model

whenever we open the browser the browser itself consider as an object and object is known as BOM.

- The another name of BOM is window
- window is the global object in frontend Javascript



* DOM / document

→ DOM :- Document object ~~ob~~ Model

whenever we execute html file in a browser the browser converts the html file to tree like structure and tree like structure is known as DOM

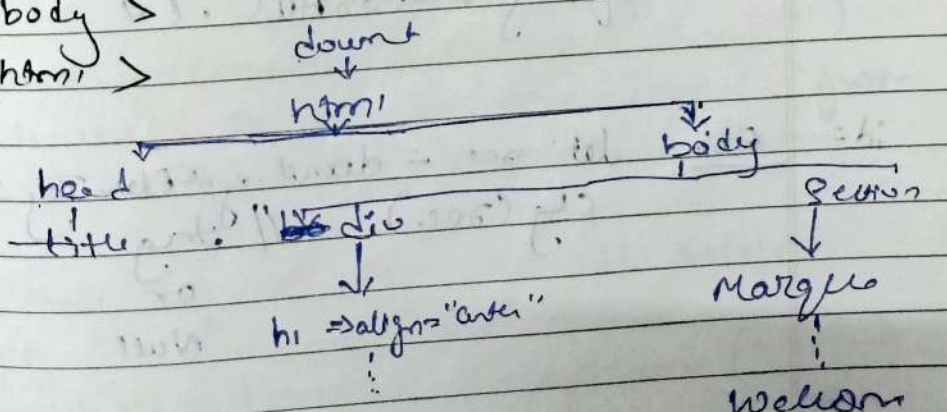
- It is a child object of BOM
- In DOM tree every html tag is treated as an object and the attribute of the tag are treated as property of the object

- Every tag is known as node of tree.

DOM :- It is used to manipulate the html dynamically through Javascript.

By using DOM we can create the html elements, attribute, delete the elements, delete the attribute, we can add the CSS, remove the CSS, add the content, delete the content and so on.

```
<html>
<head>
    <title> DOM </title>
</head>
<body>
    <div>
        <h1 align="center"> Hello </h1>
    </div>
    <Section>
        <marquee behavior="alternate direction:"right">
            welcom </marquee>
    </section>
</body>
</html>
```



Note: In the browser's developer tool the element section represents dom structure.

whatever is available in the dom that will be displayed on the UI

* Access to the html

- document
- document.head
- document.body
- document.scripts
  HTMLCollection [script]
- document.title

* DOM methods to get the ~~methods~~ elements from html

document.getElementById("")
ByTagName("")
ByClassName("classnam")
.querySelector(".#")
.querySelectorAll(".#")

id= Ex: let sec = document.getElementBy id("")
clg(sec) // tag
or
Null

i] document.getElementById(" ")

→ It is used to get the html element based on id name.

→ It accept one argument where the argument is element's id name.

→ This ~~element~~ return the first element if the given id is available, else it return <u>null</u>

ii] document.getElementsByTagName(" ")

→ It is used to get all the element from html document based on tag name.

→ It accepts one argument where the argument is the tagname, it return all the elements in the form of htmlCollection, if the given tag is available

→ else it return empty html Collection.

<u>ex:</u> let res = docum.getElemBy Tagnam ("div")
cl(res)
cl(res[0]) // accessing the first

∴ If no element found then <u>empty array</u>

iii] document.getElementByClassName(" ")

→ It is used to get all the elements from html document based on class name

→ If there is no class then it return empty

<u>ex:</u> let res = docum.getElemByClassName(" ")
cl(res) // [div ....]
or
[ ] //empty

iv) document.getQuerySelector (". / # ")

→ It is used to select the element from html document based on selector.

→ It accepts one argument where the argument is the selector name.

→ It returns first occurred element for given selector, based on selector name, else it returns null.

ex: let res = document.getQuerySelector (". test")
cl (res)

v) document.querySelectorAll (". / # ")

→ It is similar to previous one, but there is a small change in it.

→ It select all the target element.

→ It returns ~~null~~ empty node list if it not find any type.

ex: let res = document.getQuerySelectorAll (". test")
cl (res

* how to get attribute and update it.

→ <html>
<body>
<input type="text" id="one" /> <h1> Hello</h1>
</body>
</html>

→ let inp = document.getElementById("one")
clg (inp)
clg (inp.type)
clg (inp.id)
inp.type = "password" // update attribute
inp.maxlength = "10" // Add extra

or
clg (inp.getAttribute ("type"))
~~clg~~ (+:
inp.SetAttribute ("type", "password")

* How to get the content of html

let h1 = document.getElementbyTagname ("h1")
clg (h1)
clg (h1.innerhtml)
clg (h1.innertext)
clg (h1.innercontent)

∴ In innerhtml it will be taken with <html> tag

∴ In innertext it will take just text
∴ In innercontent it will get like pre tag give the space.

∴ we can add text to html from js

let section = document.getElementById("section")

section.innerHTML = "<h1> Hello <h1>"
∴ Here h1 is also pointed as text

section.textContent = "<h1> Hello <h1>"
∴ Here h1 is also pointed as text

section.innerHTML = "<h1> Hello <h1>"
∴ In this it recognise element and correct the code or update it

* Adding the CSS to the html through dom

→ way (1) inline CSS

section.style.color = "green"
section.style.fontSize = "100px"

→ way (2) inline CSS

section.style.cssText =
color : green,
font-size : 100px,
. . . .
. . . .
text-shadow : 2px 2px 4px Red

classmate
Date
Page

→ Way (3) external CSS.

. Section {
color : green,
font size : 100px
}

Section . classnam : "demo"

* Adding multiple class to an element, & toggle

→ name <p> Hello <\p>
<p id = js class = "Demo"> Hello <\p>

→ JS
let js = document.getelementbyid ("JS")
js.className = "demo - border" (It will give space as text)
// It is going to over write the prev class

js.classlist.add ("demo")
js.classlist.add ("demo - border")
js.classlist.remove ("demo - border")

∴ If we want to add multiple class to an element, we can use classlist it will add an extra class to an element.

* Toggle

It will print which is not there
∴ If 1 is there, then print 2, 2 is there then 1

js.classlist.toggle ("demo - border");

* Event handler

→ Eventhandler are the special type of attribute in html which allow us to call the function, based on user action.

→ Event → function
All the eventhandler are prefix with 'on'.

→ Type of event handler

1] Button / click Events → Onclick, ondbclick
2] MOUSE → onmouseover; onmouseout
3] Input → onfocus, onselect, onblur, onchange
4] Keyboard → onkeyup, onkeydown, onkeypress
5] Form → onSubmit.

Ex!. Button

html ⇒ <img onclick = "Show()" id = "img" />
<button onclick = Show() > Hide </button>

JS ⇒
let img = document.getElementById("img")
~~img.style~~
function Show() {
img.style.visibility = "hidden"
}

function hidden () {
image.style.visibility = "visible"
}

ex:- Mouse

<h1 id="heading" class="demo" onmouseover="test()">Hi APS

onmouseout

onmousemove

onmouseup

onmousedown

∴ we can add multiple handler in same element

Ex:- Input

<input type="text" onfocus="test()" placeholder="onfocus">

onblur =

onselect =

onchange =

onblur

onselect

onchange

∴ i] when we click in the box, it execute
ii] when we click and come out, it execute
iii] If we copy text from input, it execute
iv] If we change or update or delete, it execute

Ex:- keyboard

<input type="text" onkeypress="test()">

onkeyup

onkeydown

∴ onkeyup is better to use

* Event propogation

Event propogation is the process of handling the events.

There are 2 type of event propogation

i] event bubbling
ii] event capturing.

i] Event bubbling

It is the process of calling the inner element first then the outer most event

By default all the event handle perform event bubbling.

inner to outer

ii] Event Capturing:

Event Capturing is the process ~~the~~ of calling the outer most first and then ~~eve~~ inner event and so on.

It can be performed using 'addeventlistener' method where we have to pass true as last argument to eventlistener

ex:- for event bubbling

→ index

```
<div class = "grandparent" onclick = "grandparent()" >
    <div class = "parent" onclick = "parent" >
        <div class = "child" onclick = "child" >

        </div>
    </div>
</div>
```

∴ whenever i click on child, it will execute and then call its parent and parent will call grand parent.

// child
parent
grand parent.

→ JS

```
function grand parent () {
    alert ("grand parent")
}
function parent () {
    alert ("parent")
}
function child () {
    alert ("child")
}
```

Ex: for event Capturing.

```
let _grandparent = document.querySelector (".grandparent")
let _parent =                              (".parent")
let _child =                               (".child")

function grandparent () {
    clg ("grand parent")
}
_grandparent.addEventListener ("click", grandpar, true)

function parent () {
    clg ("parent")
}
_parent.addEventlistener ("click", parent, true)

function child () {
    clg (".child") }
_child.addEventlistener ("click", child, true)
```

```
<html>
    <head>
    </head>
    <body>
        <div>
            <Button> click </button>
        </div>
    </body>
</html
```

event Capturing

event bubbling.

classmate
Date
Page

* Define addEventListner.

It is a method which is used to add event handler to is.

It accepts 3 arguments i] handle name, without on
ii] the callback function (event)
iii] Boolean, true or false.

∴ the third argument by default false which allows event bubbling.

∴ If it is true then it is event capturing.

* Remove eventlistener.

html-> <button id="btn">greet</button>
<button> remove </button>

JS -> function greet() {
cog ("good morning")
}

let btn = document.getElementById ("btn")
btn.addeventlistner ("click", greet)

function remove() {
btn.removeEventListner ("click", greet)
cog ("removed")
}

∴ The function name should be correct dont overwrite

ex:- for password to hide and show

html ->

```
<input type="text" id="pass"
<input type="checkbox" id="toggle onclick="togglepass"
<label ~~type~~ for="toggle" > show </label>
```

JS ->

```
function togglepassword () {
  let pass = document.getElemby Id ("pass")
  if (pass.type === "password")
  {
    pass.type = "text"
  }
  else {
    pass.type = "password"
  }
```

* Timing function

There are 2 types of timing functions
i] SetTimeout
ii] SetInterval

i] SetTimeOut

SetTimeOut is a function which is used to call another function after some time "only once".

It accepts 2 arguments where the first is callback function which will be executed at the same time.

And second argument is timing function (milli seconds).

→ To stop the setTimeout we use clearTimeout function.

clearTimeout accepts one argument where the argument is the reference of setTimeout.

Syntax of setTimeout

```
setTimeout (() => {
        console.log ("Hi")

    }, 3000)
```

∴ After 3sec it will print.

ii SetInterval

→ Set interval () is a function which is used to call another function after every span of time.

It accepts two arguments the first argument is the callback function which will execute after every "span" of time and the 2nd will be in millisecond timing.

→ To stop SetInterval we use clearInterval ()

→ The clearInterval () accepts one argument where the argument is the return of SetInterval ().

Syntax for SetInterval ()

```
SetInterval (() => {
    alert ("Hello")
  }, 3000)
```

∴ After every 3sec it will execute

Ex: use setInterval and clearInterval

JS ->
```
let count = 0
let interval
function counter () {
    clg (++count)
}
let startcount = () => {
    interval = setInterval (counter, 1000);
}
function stop () {
    clearInterval (interval)
}
```

HTML ->
```
Start and Stop
```

Ex:

* Date functions

```
let demo = new Date()
clg (date)
clg (date.getFullyear())
clg (date.getMonth())
clg (date.getDay())
clg (date.getDate())
clg (date.getHour())
clg (date.getMinutes())
clg (date.getSeconds())
clg (date.getMilliseconds())
~~clg (dat~~
```

we can set the date function also.

```
date.setMonth(4)
clg (date)
```

Math methods with example

1] Math.random ( )
It generate the number betn 0 - 1

2] Math.floor (0.9) [or] Math.trunc
clg ( ) // 0 ⟷ both are same

3] Math.round (0.45) | (0.5) or above
clg ( ) // 0 | 1

4] Math.ceil (0.1) | (0.5)
clg ( ) // 1 | // 1

5] Math.abs (0.35)
It will print exact value // 0.35

Same

7] Math.sign (0.1) || 0.5
clg ( ) // 1 || 1

8] Math.max
find max value in array

9] Math.min
find min value in array

10] Math.PI
give the Pie value (3.144

11] Math.Pow
if we want to know power of some
ex:- 2^{10}
Math.pow (2, 10) // 1024

* Web Storage

The storage which are provided by the browser is known as web storage

i] Local storage
ii] Session storage
iii] Cookies

i] Local storage

→ It is used to store the data inside the browser in the form of key and value pair.

→ Size of local storage is 5MB to 10MB

→ The local storage persist (store) the data even after the browser is closed

ii] Session storage

→ It is used to store the data inside the browser in the form of key and value pair.

→ Size of local storage is 5MB to 10MB

→ It delete the data once the browser is closed

ls.length
ls.key(0)

Ex: let ls = localStorage
clg (ls) // { }
clg (ls.length) // 0

→ ls.setItem ("Name", "prasad")
ls.setAge ("Age", 25)

→ ls.getItem ("age")

→ clg (ls.key(1)) // {Name, Age}
// Age

→ ls.removeItem ("Age")
→ ls.clear () // It will delete all data

Ex: facebook

Ex:- let SS = SessionStorage
clg (SS) // { }
clg (SS.length) // 0

→ SS.setItem ("Name", "prasad")
SS.setItem ("Age", 27)

→ SS.getItem ("Age")

→ clg (SS.key(1)) // {Name, Age}
// Age
→ SS.removeItem ("Age")
→ SS.clear ()

Ex: phonepay

∴ If we switch the tab it will be expire.

JSON And API

API → Application programming Interface

- It is a media which is interacting betn User and Server.

- APIs take request from the user and give to the Server.
  Server second something that user gets.

- API is just an URL, which carries the data in the form of JSON.

JSON : Javascript Object Notation

- It is a light weight data exchange JSON format betn client and Server.

- It is similar to JS object where the key must be double quote
  ex: "Name" : "praful"

Object	JSON

let Student = {
id : 1,
name : "praful",
}
	{
	"id" : 1,
	"Name" : "John"
	}

JSON.parse() JSON.stringify()

* Conversion of JSON to object & object to JSON

ex:-
```
let Students = [ {
                  name : "Pooja",
                  id   : 101
                },
                {
                  name : "Sneha",
                  id   : 102
                } ]

clg (Students)   // [ {...}, {....} ]
```

→ let jsonStudent = JSON.Stringify (Students)
clg (jsonStudent) // {"name": "pooja" .....}

→ clg (JSON.parse (jsonStudents)) // {...},{...}

* JSON.Stringfy

It accepts 3 argument (name, replacer, space)

To avoid a particular thing use array format or callback function

```
clg (JSON.Stringfy (Students, (key, value) => {
                                  replacer
        if (key. == "id") {
            return undefined
        }
        else
        {
            return value
        }
    }, 5))
```

∴ we get only name because we have removed id

* If we want to store the data in localstr or anywhere.

Step 1 :- convert the object to stringify
Step 2 :- to get the stored data in same way, use isonparse.

localStorage.setItem("student", JSON.stringify(student))
cg (JSON.parse(localstorage.getItem("student")))

* Shallow copy and deep copy

i] Shallow copy :- It is used to change & copy value of ~~single~~ ~~level~~ few obj

ii] Deep copy :-
a] Single level hierarchy // any nested effect
b] Both will not change.

* CallStack // event loop
The event loop looks at the callback queue and the CallStack if the call Stack is empty it pushes the first item in the queue onto the stack.

* Truthy value and falsy value

true → if (true), if ({}), if ([])
if (42), if ("0"), if ("false")
if (new Date()), if (-42), if (3.14)
if (-3.14), if (infinity), if (-Infinity)

false → if (false), if (null), if (undefined),
if (0), if (-0), if (NaN),
if ("") , if (0n)

|| → short circuit operator

ex:- Assign 3 form

```
let user = {fn, mobile, pass};
let users = JSON.parse(localStorage.get("users")) || []
users.push(user)

or
if (users) {
    users.push(user)
} else {
    users = [ ]
    users.push(user)
}

localStorage.setItem("users", JSON.stringify(users))
```

* Callback hell

Callback hell → promises → Async wait

* Callback hell refers to the situation in javascript where multiple nested callbacks create complex, deeply indented code, often called the "pyramid of Doom".

- This structure makes the code difficult to read, debug and mantain, resulting in poor code quality and scalability issues.

ex:-

```
function add (val, callback) {
    return callback (val + 10, false)
}
function sub (val, callback) {
    return callback (val - 10, false)
}
function mul (val, callback) {
    return callback (val * 10, false)
}
function div (val, callback) {
    return callback (val / 10, false)
}

addition (10, (AddRes, error) => {
  if (!error){
      clg (`Addition result ${AddRes}`)
      Substraction (AddRes, (SubRes, error) => {
      if (!error) {
          clg (`Substraction ${SubRes}`)
          multiplication (SubRes, (mulRes, error) => {
```

```
if (!error) {
    clg ("multiplication # {mulRes}")
    division (mulRes , (divRes , error) => {
        if (!error) {
            clg ("division # {divRes}")
        }
    })
}
})
}
})
}
```

classmate
Date
Page

* Promises

-> It is Solution to the Callback hell

-> The Code either execute or fail in both the Cases user will get notified.

Note:- It Either execute resolve or reject

Syntax:-

```
let P = new Promise (resolve, reject) => {
  })
clg (P)
```

-> Status
Pending
fulfilled
reject

1] Pending State

```
let P = new Promise (resolve, reject) => {
  resolve
  })
clg (P)          // pending State.
```

2] fulfilled State

```
let P = new Promise (resolve, reject) => {
  resolve ("Hi")
  })
clg (P)          // promisetyp : promise
                 PromiseStat : "fulfilled"
                 promiseRaser : "hi"
```

iii] Reject State

```
let P = new Promise (resolve, reject) => {
    reject ("I am rejected")
})
clg (P)
```

// prototype : promise
promiseState : rejected
promiseResult : "hi"

promise

Static methods
i] promise.all ()
ii] promise.allSettled()
iii] promise.any ()
iv] promise.race ()

Instance method
i] .then ()
ii] .catch ()
iii] .finally()

*	Methods	Description
i]	all (i)	Wait for all promises to be resolved or any one to be rejected.
ii]	allSettled (i)	Waits until all the promise are either resolved or rejected
iii]	any (iterable)	Returns the promise value as soon as any one of the promise is fulfilled
iv]	race (iterable)	wait until any of the promise is resolved or reject.
v]	Reject (reason)	Returns a new promise object that is rejected for the given reason
vi]	resolve (value)	Returns a new promise object that is resolved with the given value
vii]	catch ()	Appends the rejection handler Callback
viii]	then ()	Appends the resolved handler Callback
ix]	finally ()	Appends a handler to the promise (Any situation it will execute)

Ex: Using then and Catch

```
let p = new Promise ((res, rej) => {
    let task = confirm ("is task completed")
    if (task) {
        res ("completed the task")
    }
    else {
        rej ("Finish the task")
    }
})
p.then ((val) => {
    clg (val)
}).catch ((err) => {
    clg (err)
})
```

* Ex: promise static methods

```
let p1 = new promise ((res, rej) => {
        res ("Resolved 1")
        rej ("Reject 1")
})
let p2 = new Promise ((res, rej) => {
        res ("Resolved 2")
        rej ("Rejected 2")
})
let p3 = new promise ((res, rej) => {
        res ("Resolved 3")
        rej ("Rejected 3")
})
```

i] promise.all ()

```
Promise.all ([P1, P2, P3]).then ((result) => {
    clg (result)
}).catch ((error) => {
    clg (error)
})
// output :- ['Result 1', 'Resolved 2', 'Result 3']
```

Note :- If any one of the promise get rejected then it will print [rejected] only. other are not executed.

ii] promise.allsettled ([P1, P2, P3])

```
Promise.allSettled ([P1, P2, P3]).then ((result) => {
    clg (result)
}).catch ((error) => {
    clg (error)
})
```

Note :- Here all will get executed but in the array of object format, either it fulfilled or Rejected.

Output :- (3) [{...}, {...}, {...}]

0 : {Status : 'fulfilled', value : "resolved 1"}
1 : {Status : 'reject', value : "rejected 2"}
2 : {Status : 'fulfilled', value : "resolved 3"}

∴ It return array of object.

iii) promise.any ()

```
promise.any([p1, p2, p3]).then((result) => {
    clg (result)
}).catch ((error) => {
    clg (error)
})
```

Note!: Here if any one also execute or fulfiled it will be displayed/output.

If ~~any~~ there is no fulfied state then we get "AggregateError" (All P is rejected)

Output!- resolved 1 (or)
resolved 2 (or)
resolved 3 (or)
AggregateError : All promise were rejected

iv) promise.race ()

```
let P1 = new promise (res, rej) => {
    settimeout (() => {
        res ("resolved 1")
        rej ("Rejected ")
    }, 4000)
})
let P2 = new Promise (res, rej) => {
    settimeout (() => {
        res ("resolved 2")
        rej ("Rejected 2")
    }, 2000)
})
```

```
promise.race ([p1, p2]).then (result) => {
    clg (result)
}).catch (error) => {
    clg (error)
})
```

<u>Output</u> :- // resolve 2

<u>Note</u>:- The one have execute first it will be the output

* <u>Promise chaining</u>

Javascript promise chaining pattern that chains the promises to execute asynchronous operation in sequence.

<u>ex</u>:-

```
let p = new Promise (res, rej) => {
    res (10)
})
p.then ((result) => {
    clg (result)
    return result + 10
}).then ((result) => {
    clg (result)
    return result + 10
}).then ((result) => {
    clg (result)
})
```

* Fetch API

Fetch api makes it easy to get information from a website and do something with that data in your browser.

Fetch api is used to get data from the internet.

i] GET. ii] POST iii] PUT iv] PATCH v] DELETE

Ex:- fetch api using promises.

∴ If we are using the promise to fetch the data then use 2 .then

```
let p = fetch("https://jsonplaceholder.typicode.com/posts/1")
p.then((value1) => {
        clg (value 1)
        return value1.json()
    })
.then ((value2) => {
        clg (value 2)
    })
.Catch ( (error) => {
        clg (error)
    })
```

∴ Here we 2 .then because one is used to convert raw data to object form.

ex :- fetch api using - Async Await

∴ Create a async function inside that we await.

```
async function fetchAPi ( ) {

    let data = await fetch ("        ")
    let finaldata = await data.json()
    cl.g (final data)

}
fetchAPi ( )
```

"https://api.github.com/users"
fake Store api
OMDb Api / TMDB / news api

* Async and Await

In JS async and await are the keywords that help make asynchronous code more readable and easier to handle.

A async function it can use promise in it.

ex:-
```
async function weather () {
    let weatherofBangalor = new promise ((res, rej) => {
        setTimeout ( () => {
            resolve ("27 deg")
        }, 3000)
    })

    let Bang = await weatherofBangalor
    return Bang
}
let a = weather()
a.then ((value) => {
    clg (value)
})
```

→ Async ⇒ The async keyword in Javascript is used to define an asynchronous function which allows you to work with asynchronous operations in a more readable way.

→ Await :- The await keyword can be used inside the function to pause its execution until a promise is resolved or rejected.

7] use strict

→ In JS 'use strict' is a directive introduced in ECMA Script 5 (ES5) to enforce a stricter or safer mode while writing JS code.

It helps catch common mistakes and make the code more secure by preventing certain actions that are considered bad practice.

ex.
```
'use strict'
x = 10        // error x is not defined
```

- helps identify bugs early during development.
- Improve performance by allowing browsers to optimize code better.

8] Define event loop

The event loop in JavaScript manages asynchronous task by continuously checking the call stack for execution and processing task from the callback queue when the stack is empty.

* BOM types and its use / windows

-> It is a collection of objects provided by the browser that interact with the browser window and its components.

i] window object
- It is the top level object of the BOM
- It represents browser window or tab
  - alert, confirm, prompt, open, close

ii] Document object
- It is known as DOM (Document Object Model)
  -> window.document

iii] Navigator object
- It Represents information about the browser.
-> navigator.userAgent : Returns the user-agent string
-> navigator.language : Returns the browser language
-> navigator.onLine : checks if the browser is online

iv] Screen object
- It provides information about the screen
-> screen.width -> in px
-> screen.height -> in px
-> screen.availWidth -> Available screen width
-> screen.availHeight -> Available screen height

v] History object
- used to navigate through the browser history
-> history.back() -> moves back one step in history
-> history.forward() -> move forward
-> history.go(n) -> moves n steps

V: location object

- Represents the URL of current page and allows navigation.

→ location.href : Gets/sets URL of current page
→ location.reload() : Reload the current page
→ location.replace() : Replace the current page

* Append child

The appendchild method is used to add a new child node (element) to a parent node. It always add the child at the last child of the parent node.

Syntax

ParentNode.appendChild(newchild)

ex:-

```
let newele = document.createElement('div')
newele.textContent = 'Hello, world'

document.body.appendchild(newele)
```

→ Remove child

ex:- let remove = document.getElementById('ele')

remove.parentNode.removeChild(remove)

i] CreateElement

It is used to create element dynamically

ex:-

```
let newDiv = document.createElement('div')
newDiv.textContent = 'Dynamic'
document.body.appendChild(newDiv)
```

ii] InsertBefore

This method add a new child before a specific child of a parent

```
let parent = document.getElementById('parent')
let newChild = document.createElement('span')
newChild.textContent = 'first'

parent.insertBefore(newChild, parent.firstChild)
```

iii] Replace child

The replace child method replace an existing child with a new child.

```
let parent = document.getElementById('parent')
let oldChild = document.getElementById('oldChild')
let newChild = document.~~getElementById~~ createElement('div')

newChild.textContent = 'It was replaced'

parent.replaceChild(newChild, oldChild)
```

iv) clone node

The cloneNode method Create a Copy of a node

```
let original = document.getElementById('elo')
let clone = original.cloneNode(true)
document.body.appendChild(clone)
```

* Event deligation

Event deligation is a technique in Javascript where a single event listener is attached to a parent element instead of attaching event listener to multiple child elements.

ex:-

```
<div onclick="demo()">
    <button> click 1 </button>
    <button> click 2 </button>
</div>
```