19/6/23

Java Script

* Java Script is use to give functionality to static web pages.

* Java Script was introduced by "Brendan Eich" in the year 1995, he was working at Netscape Navigator.

* Java Script is a world's most popular scripting language, it is a light weight interpreted language which is used by several websites for scripting the web pages.

* Java Script is a client-side scripting language because it runs on ~~the~~ client side or client machine inside the web browser.

* Initially javascript name changed several times.

* The first name is given to javascript is "Mocha"

* In the Netscape Navigator in sept 1995 it got new name called "Live Script".

* In Netscape Navigator in dec 1995 its got its final name as Java Script.

* The ECMA from the organisation (European Computer Manufactor Association) they took over the java script and then hoist the primary standards & implemented new technologies & started calling as ECMA script.

* The current version of java script is ECMA script 6.

Difference b/w Java & JavaScript.

Java	JavaScript
* It is a programming language	* It is a scripting language.
* It is a multi-threaded language.	* It is single-threaded language.
* It is a strictly-typed language.	* It is weekly-typed language.
* It runs on JVM.	* It runs on all the browsers.
* More memory used in java.	* Less memory is used in java script.
* It is a independent language.	* It is a dependent language which is executed along with html.
* It is both compiled and interpreted language.	* It is interpreted language.
* It is a low-level language.	* It is a high-level language.
* multi call stack	* single call stack

* Java Script is a high-level language which gets executed on the browser.
* Browser is a application which behaves like client.
* Browser itself acts like a compiler or interpreter.
  → Browser gives an environment for JS to run
  Browser has a sub application called as JS engine.

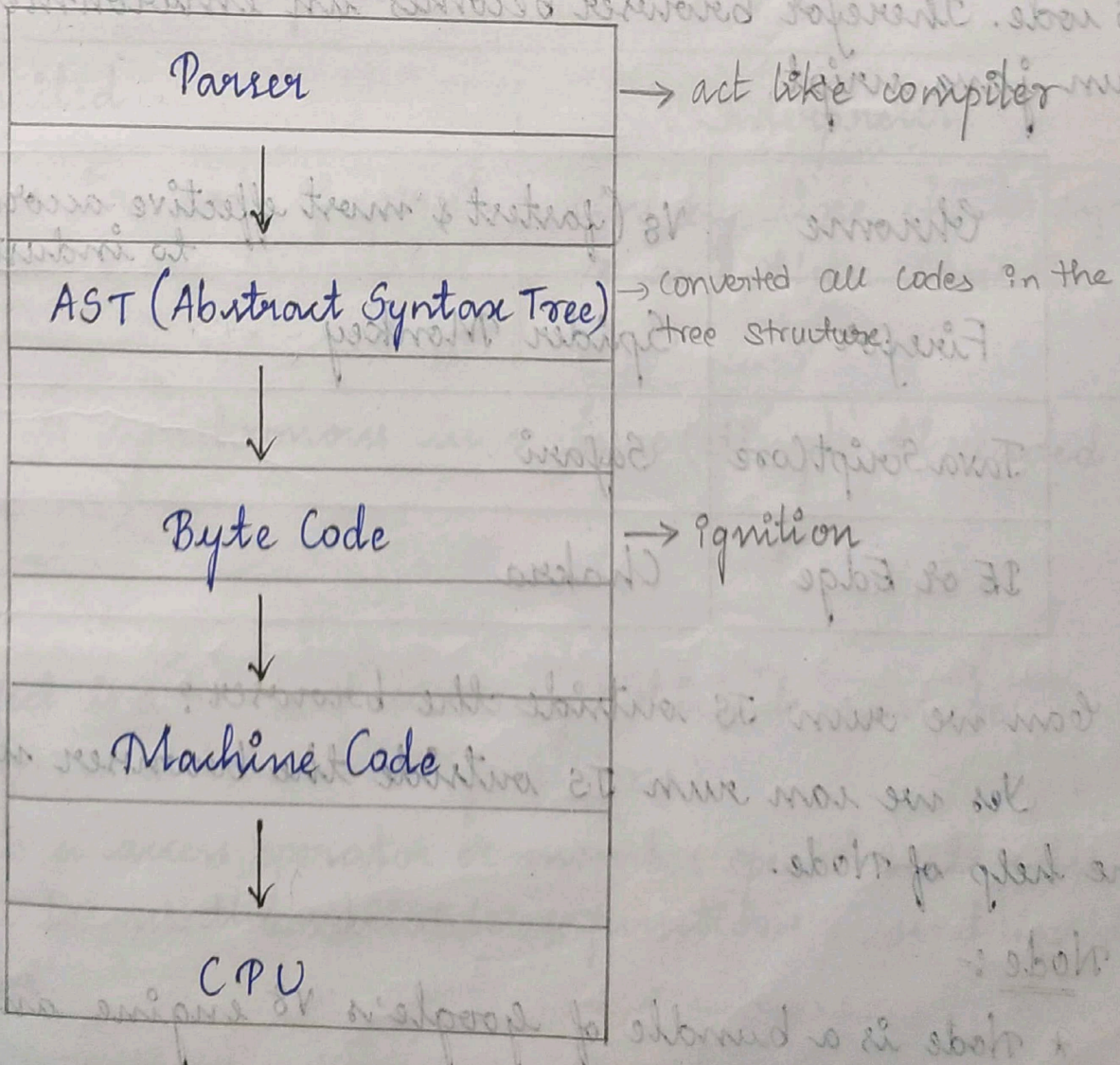
⇒ JS engine is responsible to translate JS instructions into machine understandable language within the browser.

⇒ We need JS engine to execute JS.

* Browser and its sub applications
  * JS Engine
  * Timer
  * Console
  * Datatypes.

Java Script code can run both inside and outside the browser.

1) Inside the browser (JS engine)



2) Outside the browser :-

Chrome → V8 engine
Mozilla → Spider monkey
Microsoft → Chakra
Safari → Core JS
Brave → Blink

* Java Script is a high level language.
* To execute JS instructions it is mandatory to translate JS instruction into machine understandable language.
* This job is done by JS Engine.
* Every browser will have JS Engine to run the JS code. Therefor browser becomes an environment to run java script.

Chrome	V8 (fastest & most effective according to industry)
Firefox	Spider Monkey
JavaScriptCore	Safari
IE or Edge	Chakra

Can we run JS outside the browser?

Yes we can run JS outside the browser with the help of Node.

Node :-

* Node is a bundle of google's V8 engine and built in methods using C++.

* It serves as an environment to run JS outside the browser.

* This invention helped JS to gain its popularity in useage as a back end language.

* Node is an enviroment to execute JS without the help of a browser.

20/6/23

Characteristics of JS :-

* Purely object oriented.
* Interpreted Language.
* JS uses Just In Time compiler (combination of compiler and interpreter).

Compiled	Interpreter
Checks entire code if correct. It generates an executable file.	Line by line checks synta -x and then execution.

* JS is synchronous in nature (Single threaded architecture)

Object :-

* Object is a block of memory which has states (variables) and behaviours (functions).

* . is a access operator or member operator. It will help you to use the variables & functions present inside an object.

Console log()

Object Operator

Console	.	log()
Object	Operator	Function of console object

Log() is an function which can accepts arguement (arguement is an data passed to a function).

First code :-

```
<html>
    <head>
        <title> Doument</title>
    <head>
    <body>
        <h1> Program</h1>
    <script>
        console.log("hello world");
    </script>
    </body>
</html>
```

Use <script src = "path"></script> to include all JS files.

To Execute JS On Browser

1) We can execute JS instructions directly in the console provided by the browser.

2) We can execute JS instructions by embedding it in html page.

a) Internal

b) External

a) Internal :-

With the help of script tag we can embedded JS instructions.

Syntax :-

```
html code ....
<script>
    JS code ...
</script>
html code .....
```

b) External :-

We can create a separate file for JS with extensions .js.

Link the js file with html file using the src attribute of script tag.

Syntax :-

```
Html code....
<script src="path/filename.js"></script>
Html code....
```

Token :-

Smallest unit of any programming language is known as Tokens.

1) Keyword :-

A predefined reserved word which is understandable by your JS engine is known as

NOTE :-

* every keyword must be in lower case.
* a keyword cannot be used as an identifier.

eg :- keyword

if, for, let, null, null

2) Identifiers :-

The name given to the components of JS like variables, functions, class, etc are known as identifiers.

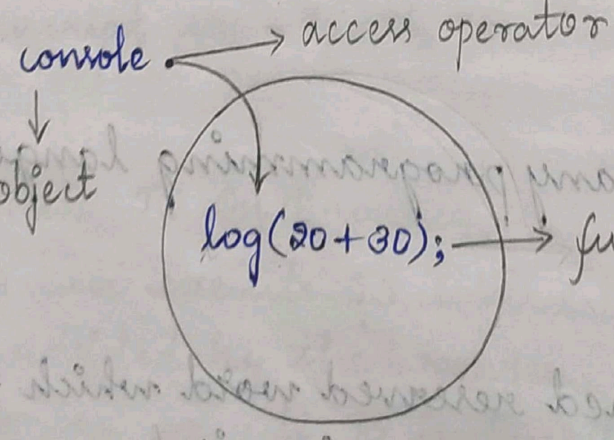
Rules for identifiers :-

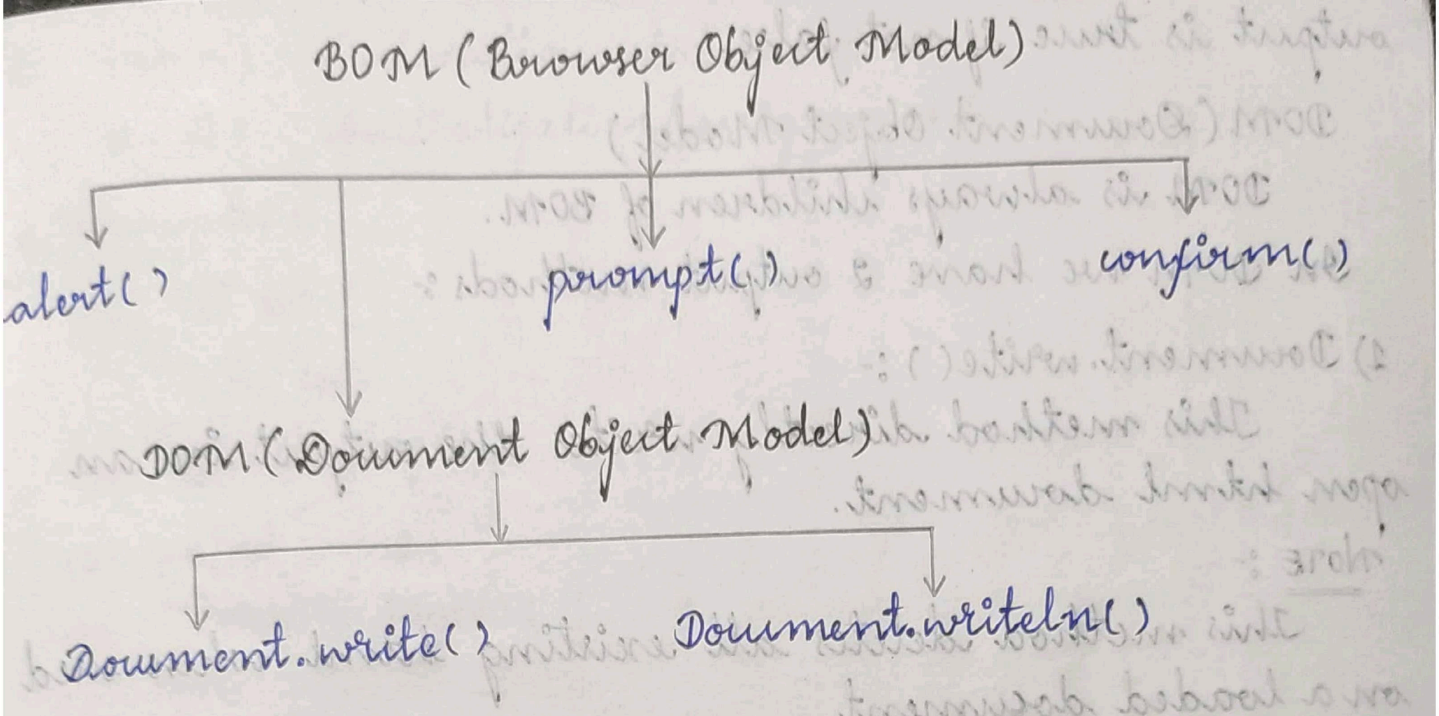
* An identifier cannot start with a number.
* Except $ & _ no other special character is allowed.
* we cannot use keywords as identifiers.

3) Literals :-

The data which is used in the javascript program is called as values or literals.

Output Methods :-





1) alert( ) :-

This method displays an alert box or pop-up with a message and an OK button.

NOTE :-

The alert box takes the focus away from the current window and forces the user to read the message, it prevents the user from accessing other parts of the page until alert box is closed.

2) prompt( ) :-

* This method displays dialogue box that prompts the user for an input.
* The method returns the input values if the user click on OK button or it returns null.

NOTE :-

When a prompt box pop-up the user will have to click either OK or Cancel to proceed.

3) confirm( ) :-

This method displays a dialogue box with a message and OK button and cancel button, this method returns the output in boolean format if we click on OK the

output is true if not false.

DOM (Document Object Model)

DOM is always children of BOM.

In DOM we have 2 output methods :-

1) Document.write( ) :-

This method directly writes the output in an open html document.

NOTE :-

This method deletes all existing html when used on a loaded document.

2) Document.writeln( ) :-

This method gives an addition writing a new li character or space after each statement.

21/6/23

Variables :-

A named block of memory which is used to store a value is known as variable (container to store data).

NOTE :-

In JS variables are not strictly typed it dynamically typed. Therefore it is not necessary to specific type of data during variable declaration.

* In a variable we can store any type of value.
* It is mandatory to declare variable before using
* Syntax for variable declaration :

```
var / let / const identifier ;
var identifier ;
let identifier ;
const identifier ;
```

23/6/23

Datatypes :-

Primitive	Non-Primitive
(1) String	(1) Function
(2) Number	(2) Arrays
(3) Undefined	(3) Object
(4) Boolean	a) date
(5) Null	b) time
(6) Bigint	c) math
(7) Symbol	

Datatypes	Values
Number	→ Any mathematical number.
String	→ Anthing enclosed in (' ', " ", ` `)
Boolean	→ False, true
Null	
Undefined	→ it is a keyword and a value.
Object	
Bigint	→ suffix the number with n.
Symbol	

* According to JS all the types except object is considered as Primitive types.
* Object is considered as non-primitive type.

NOTE :-

* The number b/w the range $-(2^{53}-1)$ & $(2^{53}-1)$ is considered as number type.

* We can use bigint type to store numbers beyond the given range. By suffixing the number n.

ex:-

1 → number

1n → bigint

*** Typeof Operator :-

* It is a keyword used as an unary operator, to identify the type of data.

Syntax :- typeof value

ex:-

```
console.log(typeof "done with file structure"); // string
console.log(typeof 47); // number
console.log(typeof true); // boolean
console.log(typeof false); // boolean
console.log(typeof null); // object
console.log(typeof undefined); // undefined
console.log(typeof 1n); // bigint
```

v.imp * The typeof null is never null it is considered as object.

(typeof console) → object

Hoisting :-

Hoisting is a JS mechanism where variables, functional declaration are moved to the top of the scope before code execution.

JS can only hoist declarations, hoisting can be achieved in var keyword not in let & const keyword.

What is Temporal Dead Zone (TDZ):-

* It is a behaviour in JS that occurs when declaring the variable with the let & const keywords but not with var.

* In ECMA script 6 accessing a let or const keywords before its declaration causes a reference error. That time span when that happens b/w the creation of ~~variatio~~ variable binding and its declaration is called as TDZ.

24/6/23

Global Execution Context (GEC):-

It is the default or first execution context i.e, created by the JS engine before any code is executed. (when the JS file first loads in the browser).

* All the global code i.e, not inside a function or object will be executed inside GEC.

* Since JS engine is a single threaded there ∅ will be only one global environment & there will be only one GEC.

GEC consists of two phases

(1) Variable phase
(2) Execution phase

1) Variable phase :-

* The code which is having all the variables will be stored or executed in variable phase.

* Initially all the variables will be initialized with undefined.

2) Execution phase :-

All the execution statement are stored or executed in the second phase i.e, execution phase.

```
let x = 300;
const y = 1000;
console.log(x);
console.log(y);
var d = 300;
console.log(d);
console.log(a);
var a = 2000;
var a = 5000;
console.log(a);
const z = 10;
console.log(z);
console.log(e);
let e = 20;
```

GEC

Variable phase	Execution phase	Output
x = ~~undefined~~ 300	c.log(x)	300
y = ~~undefined~~ 1000	c.log(y)	1000
d = ~~undefined~~ 300	c.log(d)	300
a = ~~undefined~~ 2000	c.log(a)	undefined
~~a = undefined~~ 5000		5000
z = undefined	c.log(z)	10
e = undefined	c.log(e)	TDZ(error)

27/6/23

String :-

Java script can be represented using ' ', " ", and ` ` (back tick).

NOTE :-

* The start and end of string must be done with the help of same quote (' ', " ", ``` ```) these are not allowed.
* If a text contains single quote then it can be represented using " ".

Back Tick (` `) :-

Back tick can be a multiline string.

ex :-
```
var str = `My Output ${20 + 30}`
console.log(str);
```

* String represented using backtick is also known as "Template string".

Advantages :-

We can substitute an expression in a string by using interpolation (${ }).

There are few string methods.

1) concat
2) charAt
3) charCodeAt
4) replace
5) ~~trot~~ indexOf
6) repeat
7) search
8) slice
9) split
10) padstart
11) padend
12) trimstart
13) trimend
14) includes
15) length
16) match
17) matchAll

Type Casting / Type Coersion :-

In JS all non-zero numbers are considered as true whereas number zero, null, non-empty string, undefined all these values are considered as false when they are converted to boolean.

Types of Type Casting :-

(1) Implicit (2) Explicit

(1) Implicit :- Process of converting one type data to another type of data by JS engine implicitly / internal. Wrong type of data is used in the expression called implicit type casting.

ex :-
```
console.log(5 - '30');
```

(2) Explicit :- The process of converting one type of value to another type of value is known as explicit type casting.

ex :-
```
console.log(number("1234"));
```

Number Methods in JS :-

(1) ParseInt
(2) ParseFloat
(3) IsInteger
(4) ToPrecision
(5) ToFixed

1) ParseInt :- ParseInt parses its arguement and return whole number and neglectes the decimal values.

ex :-
```
var a = 100.02;
var b = parseInt(a);
console.log(a); // 100
```

2) ParseFloat :- ParseFloat parses its arguements and returns floating point number i.e, the decimal values.

ex:- var num = 300.354;
var a = parseFloat(num);
c.log(a); // .354

3) ToFixed :- Tofixed returns a string with the number written with specified number of decimals.

ex:- var num = 300.546;
var a = num.tofixed(2);
c.log(a); // 300.55.

4) ToPrecision :- It returns a string with a number written with the specified length.

ex:- var num = 300.543;
var a = num.toprecision(3)
c.log(a); //

28|6|23

Functions :-

(1) Anonymous function
(2) Named function
(3) Function with expression
(4) First order function.
(5) Immediate invoking function
(6) Arrow function
(7) Higher order function
(8) Nested function
(9) Callback function
(10) Generator function.

5) IsInteger :- It returns whether the value is integer or not, isIntger will always passes the output in boolean format.

ex:- var num = 354;
var a = number.isInteger(num); c.log(a); // true.

* Function is a block of instructions which is used to perform specific task.

* Function gets executed only whenever it is called or invoked.

* The main advantage of function we can achieve code reusability.

Syntax :-

```
function name( )
{
   Block of statements ;
}
Name( ) → function call
```

Generally we can create a function in 2 ways

(1) using function declaration statement (named function)

(2) function expression

There are 10 types of functions

1) Anonymous function :-

A function without a function name we call it as anonymous function.

Syntax :-

```
function ( )
{

}
( )
```

2) Named function :-

A function with name we call it as named function

Syntax :-

```
function x( )
{
  console.log("Puppy");
}
x( )
```

3) Function with expression :-

Passing a function as an value to the variabl we call it as function with expression

Syntax :-

```
var a = function ( )
        {
          console.log(20+30);
        }
        a( )
```

This Keyword :-

* It is a keyword used a variable.
* It holds the address of global window object.
* Therefore with the help of this variable we can use the members of global window object.
* In JS, this is a property of every function. (every function will have this keyword).
* Used to access the members of global execution

```
var a = 10;
function test()
{
  var a=20;
  function demo()
  {
    console.log("demo");
  }
  console.log(a); //20
  console.log(this.a); //10
}
test();
```

29/6/23

Function Execution Context (FEC):-

Whenever a function is invoked a javascript creates different type of execution context is known as FEC.

* within the global execution context to ~~execute~~ evaluate and execute the code within that function.

```
var a = 20+30;
let b = 3000;
const c = "Magic";
console.log(c);
console.log(a,b);
function rex()
{
  console.log(this.a);
  var x = 3000;
  let y = 5000;
```

```
    console.log(x);
}
rex();
console.log(y);
```

GEC

variable phase	execution phase
a ud 50 b ud 3000 c ud Magic rex()	clog(c) clog(a,b) clog(y) error

output :-

Magic
50, 3000
error

FEC

variable phase	execution phase
this : window x ud 3000 y ud 5000	clog(this.a) clog(x)

output :-

50
3000

30/6/23

4) Immediate Invoking Function Expression IIFE :-

When the function is called immediately as soon as the function object is created it is known as immediate invoking function.

Steps to achieve it

(1) Treat a function like an expression by declaring it inside a pair of brackets.

(2) Add another pair of brackets next to it which behaves like a function call statement.

ex :-

```
(function abc()
{
    console.log("Hello");
}
());
```

1/7/23

Return Type / Return Keyword :-

* It is a keyword used as controlled transfer statement in function.
* Return will stop the execution of a function & transfers the control along with data to the caller.

ex :-

```
function toMeter(cm)
{
  return cm/100;
}
var cms = 2546;
console.log(toMeter(cms));
var m = toMeter(cms);
console.log(m);
```

5) Arrow Function :-

* Arrow function was introduced ES6 version of java script.
* Main purpose of an arrow function is to reduce syntax.
* The arrow function is also called as "Lambda functions", in arrow function we donot use function keyword instead we go with "⟹ fat arrow."

* we can use this method when we want our code to be short and not call the function each time.

Rules :-

1) Parameters is optional.

2) If a function have only one statement then the block (braces) is optional.

3) If a function has more than one statement it is mandatory to create a block.

4) If we pass return keyword in a single statement than block is always mandatory.

Scope Chaining [Lexical Scope] :-

The ability of js engine to search for a variable in the outer scope when it is not available in the local scope is known as scope chaining.

The scope chaining is generally established b/w

1) A function and the global object

ex :-

```
let a = 10;
function test()
{
  a++;
  console.log(a);
}
test();
```

6) Higher Order Function :-

It is a function which accepts another function as parameter.

7) Call Back Function :-

It is a function which is passed as a argument to an another function.

```
function HOF (call back)
{
  callback( )
    c.log("Iam a HOF accepts another function"
}
HOF(Demo)
{
  function demo( )
  {
    c.log("Iam a demo function acting like a
          call back");
  }
```

output :-

I am a demo function acting like a call back
I am a HOF accepts another function.

8) First Order Function :-

It is a function which can be stored as a value inside the variable.

Syntax :-

```
var demo = function test( )
{
```

```
    c.log("I am first order function stored inside
                the variable");
}
demo();
```

9) Nested Function :-

In nested function we can define a function inside another function.

NOTE :- The inner function (child function) is local to the outer function (parent function)

ex :-

```
var a = 12;
c.log(10);
function parent()
{
  var name1 = "Raj"
  c.log(name1)
  function child()
  {
    var name2 = "Raj's son"
    c.log(name2);
  }
  child()
}
var b = 10;
parent();
c.log(10);
```

output :-

```
12
260
Raj
Raj's son
10
```

Closure :-

It is a function which holds the reference of the variable from the previous function even after the function has executed and removed/return from the call stack.

Closure is created only for the variable which is utilised by ~~or~~ any child function.

Function can return a function.

```
function test( )
{
  c.log(" I am a test function ");
  function demo( )
   {
     c.log(" I am a demo function ");
   }
return demo
}
test( )( )
```

Program for Lexical scope and closure

```
let bmtcbus = " blue bus ";
function parent( )
{
  var car = " audi "
  var ajji = " xyz "
  c.log(car);
 function child( )
 {
   var car1 = " alto "
   c.log(car1)
```

```
function gc()
{
  var car2 = "second handled"
  c.log(car2);
  c.log(car);
  c.log(bmtcbus);
}
return gc
}
return child
}
parent()()()
```

output :-

audi
alto
second handled
audi
bmtcbus

Array :-

* Array is non-primitive datatype in JS.
* It is a linear data structure used to store the multiple elements based on the index values.
* The index value starts from 'zero' (0).
* In JS array we can store heterogeneous type of data.
* Size is dynamic its not fixed.

NOTE :-

Type of Array = is a object

```
console.log(typeof Array);
```

output :- object

ex :-

```
var arr = [ ]
    arr[0] = 12;
    arr[1] = "Hi";
```

```
arr[3] = undefined
arr[4] = null
arr[5] = 12n
arr[6] = {c.log("I am a function");}
arr[7] = ["html", "css", "js"]
c.log(arr);
c.log(typeof arr);  output :- object
c.log(Array.isArray);  output :- true
```

Array Methods :-

push( )	sort( )	every( )
pop( )	forEach( )	indexof( )
unshift( )	map( )	lastIndexof( )
shift( )	filter( )	includes( )
slice( )	reduce( )	concat( )
splice( )	join( )	
reverse( )	some( )	

1) push( ) :-

This method adds the new elements to the array from the last.

ex :-
```
var arr = [0,1,2,3,4,5]
arr.push(6,7);
c.log(arr);
```
output :- [0,1,2,3,4,5,6,7]

2) pop( ) :-

This method will remove one element from the last of the array.

ex :-
```
var arr = [0,1,2,3,4,5]
arr.pop( );
```

```
c.log(arr);
```

output :- [0, 1, 2, 3, 4]

3) unshift() :-

This method add the new elements to the array from the starting of the array.

ex :-
```
var arr = [0, 1, 2, 3, 4, 5]
arr.unshift(7,8);
c.log(arr);
```

output :- [7, 8, 0, 1, 2, 3, 4, 5]

4) shift() :-

This method will remove 1 element from the start of the array.

ex :-
```
var arr = [0, 1, 2, 3, 4, 5]
arr.shift()
c.log(arr);
```

output :- [1, 2, 3, 4, 5]

5) slice() :-

This method is used to fetch the array elements based on the index values.

NOTE :-

* we can fetch elements based upon negative index values.
* Slice() will not affect the original array.

ex :-
```
var arr = [0, 1, 2, 3, 4, 5]
c.log(arr.slice(2, 4));
c.log(arr.slice(-4, -2));
c.log(arr);
```

output :- [2, 3]
[2, 3]
[0, 1, 2, 3, 4, 5]

6) splice( ) :-

This method is used to add the elements or delete them from the array.

* This method will affect the original array
* This method accepts multiple arguements

1) starting indese value → 1st value
2) count of the elements to be removed → 2nd arg
3) the new elements to be added → 3rd arg

Syntax :-

```
arr.splice(start, count, new, new)
```

ex :-
```
var arr = [0, 1, 2, 3, 4, 5]
arr.splice(6, 0, 6, 7)
arr.splice(3, 2)
arr.splice(3, 2, 30, 40, 50)
c.log(arr);
```

output :- [0, 1, 2, 3, 4, 5, 6, 7]
[0, 1, 2, 5]
[0, 1, 2, 30, 40, 50, 5]

7) reverse( ) :-

This method will reverse the order of elements in an array.

ex :-
```
var age = [12, 10, 67, 100, 94, 34]
c.log(age.reverse( ));
```

output :- [34, 94, 100, 67, 10, 12]

8) sort() :-

This method sorts the element in an ascending order considering elements as a string.

NOTE :-

If we want to sort numerical value pass an callback function which accepts two parameters to the sort() method.

ex :- var arr = ["apple", "mango", "orange", "banana", "grapes"]

c. log(arr. sort());

output :- ["apple", "banana", "grapes", "mango", "orange"]

var age = [12, 10, 67, 100, 94, 34]

c. log(age);

c. log(age. sort((a, b) => a-b)); // ascending

c. log(age. sort((a, b) => b-a)); // descending.

output :- [10, 12, 34, 67, 90, 100]

[100, 90, 67, 34, 12, 10]

9) forEach() :-

This method is inbuilt higher order function which accepts another call back function as its parameter.

* This method will iterate over each elements of the array for the given callback function.
* It does not support return value.

ex :- var age = [12, 10, 67, 100, 90, 34]

var ans = age. forEach((item) => {c. log(item + "100rs given")})

c.log(ans);

output :- 12 100 rs given
10 100 rs given
67 100 rs given
100 100 rs given
90 100 rs given
34 100 rs given
undefined.

10) map() :-

This method is a inbuilt higher order method which accepts another call back function as its parameter.

* This method will iterate over each element of the array for the given call back function.
* The return value from the callback function for each element will be added into the new array.
* The map() method will return backs the new array.

ex :- var age = [12, 10, 67, 100, 90, 34]

c.log(age)

var ans = age.map((item) => {return item + "100 rs given"})

c.log(ans);

output :- [12, 10, 67, 100, 90, 34]

['12 100 rs given', '10 100 rs given', '67 100 rs given', '100 rs given', '90 100 rs given', '34 100 rs given']

11) filter() :-

This method is a inbuilt higher order function which accepts another callback function as its

parameters.

* This method is used to iterate over each elements of the array and the provided callback function will checks for the condition.

* If the callback function returns true then that element will be added to the array.

* If returns false then the element will not be added to the array.

ex:-
```
var age = [12, 10, 67, 100, 90, 34]
    c.log(age);
var ans = age.filter((item) => {(return item > 87)})
    c.log(ans);
```

output:-
```
[12, 10, 67, 100, 90, 34]
[67, 100, 90, 34]
```

12) Reduce ():-

This method will iterate over each element of the array with the provided callback function.

* The callback function accepts two (2) parameters 1st = initial value, 2nd = current element.

* The return method will return single value.

ex:-
```
var age = [12, 10, 67, 100, 90, 34]
    c.log(age);
var ans = age.reduce((iv, cv) => {(return iv + cv);})
    c.log(ans);
```

output:-
```
[12, 10, 67, 100, 90, 34]
```

13) join( ) :-

This method will convert an array into a string with specified separator.

ex :- var age = [12, 10, 67, 100, 90, 34]

c.log(age.join(" "));

output :- 12 10 67 100 90 34

14) some( ) :-

This method will iterate over each element of the array.

* Each element has been tested with the given condition or case.

* If any one element satisfies the condition then the some( ) method will return true value.

ex :- var age = [12, 80, 60, 50, 90, 17]

c.log(age.some((elem) => {return elem >= 18}));

output :- true.

15) every( ) :-

If any one element fails the condition then the every( ) method will return false. value.

ex :- var age = [12, 80, 60, 50, 90, 70]

c.log(age.every((elem) => {return elem >= 18}));

output :- false

16) includes( ) :-

It checks whether the element present or not.

ex :- var age = [12, 80, 60, 50, 90, 70]

c.log(age.includes(80));

output :- true

17) indexof( ) :-

It fetchs the starting indexe value based on the elements given.

ex :- var age = [12, 80, 60, 50, 80, 90, 70]

c.log(age.indexof(80));

output :- 1

18) lastindexof( ) :-

It fetches the last indexe value based on element given.

ex :- var age = [12, 80, 60, 50, 80, 90, 70]

c.log(age.lastIndexof(80));

output :- 4.

19) concat( ) :-

This method adds the neret element.

ex :- var age = [12, 80, 90, 60, 50, 70]

var two = ["hi", "hello"]

c.log(age.concat(two));

output :- [12, 80, 90, 60, 50, 70, 'hi', 'hello']

10/7/23

Object :-

* Object is a non-primitive datatype.
* Object is use to store the properties of the real time entity in the form of key and value.
* Objects are mutable.

Objects can be created in 3 ways

1) Literal way :-

```
var emp = {
    name : "Raj" ;
    id : 123 ;
  }
```

2) By using object constructor

```
var emp = new Object( )
emp.name1 = "raju" ;
emp.id = 123 ;
console.log(emp) ;
```

3) Constructor function

```
function Employee (name1.id)
{
  this.name1 = name1 ,
  this.id = id
}
var emp1 = new Employee("raju", 123)
var emp2 = new Employee("rani", 456)
  console.log(emp1) ;
  console.log(emp2) ;
```

Object.Freeze( ) :-

This method freezes the object and prevents the object from being modified.

```
var emp = {
    name1 : "Raj",
    id : 123,
    skills : ["html", "dsa", "js", "react"],
    student : true
```

```
    job:"developer"
}
Object.freeze(emp)
emp.name1 = "rani";
delete emp.id;
emp.sal = "12 lpa";
console.log(emp);
```

Object.isFrozen( ):-

* This checks / determines whether the object is frozen or not.
* This method returns back boolean value.

```
console.log(Object.isFrozen(emp));
```

Object.Seal( ):-

* This method prevents the object from being added with new properties or deleting the existing properties.
* We can only update the value of the existing property.

```
Object.seal(emp)
emp.name1 = "rani";
delete emp.id;
emp.sal = "12 lpa";
console.log(emp);
```

Object.isSealed( ):-

* This method checks/determines whether the object is sealed or not.

```
for(let x in emp)
{
  c.log(x); // it will returns keys
}
```

output :-

name1
id
skills
student
job

```
for(let x in emp)
{
  c.log(emp(x)); // it will returns values
}
```

2) for of loop / (for ... of) :-

for of loop is use to iterate over the values of the any iterable object.

```
for(let x in emp)
{
  c.log(x);
}
```

output :-

raj
123
["html", "css", "dsa"]
true
developer

Class :-

* Classes are introduced ES6 version of js.
* js classes are similar to the javascript constructor function.
* The class keyword is use to create a class.

```
class Student {
    constructor(name1, id)
    {
      this.name1 = name1,
      this.id = id
    }
}
var std1 = new Student("raju", 123);
var std2 = new Student("raj", 456);
console.log(std1);
console.log(std2);
```

12/7/23

Object destructuring :-

It is an expression use to extract properties from the object and assigning to the variable.

```
var student = {
      name1 : "Raju",
      id : "123",
      add : {
          road : "8th cross"
      }
}
console.log(student.name1);
let {name1, add} = student
console.log(student.add.road);
console.log(add.road);
```

Math Object :-

It is a inbuilt object which has many properties and methods, which can be used for the mathematical operations.

```
console.log(Math.PI);
console.log(Math.sqrt(64));
console.log(Math.cbrt(27));
console.log(Math.sin(0));
console.log(Math.cos(0));
console.log(Math.tan(0));
```

Loosely equal to

compares only the values

ex :-
```
console.log(12==11); // false
console.log(12==12); // true
```

Strictly equal to

compares values as well as datatype

ex :-
```
console.log(12===12); // true
console.log(12==="12"); // false
```

Math.Random() :-

```
var otp = (parseInt((Math.random()*10000)));
alert("ur otp num is ${otp}")
var verify = Number(prompt("enter ur otp num"));
if(otp==verify)
{
```

```
        console.log("Logged in Successfully");
    }
    else
    {
        console.log("otp mismatched");
    }
```

⇒

```
var ans = 10/3;
console.log(Math.floor(ans));
console.log(Math.ceil(ans));
console.log(Math.round(ans));
console.log(Math.round(10.4));
console.log(Math.min(12,13,16,98,76,1000));
console.log(Math.max(12,13,16,98,76,1000));
console.log(Math.pow(2,3));
```

Date Object :-

It is a inbuilt object which has some methods which is use to get or set the date and time.

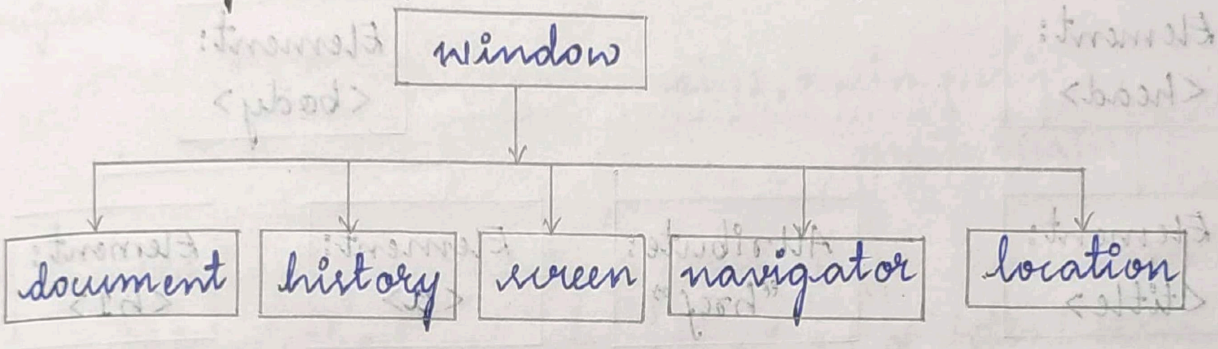
```
var ref = new Date()
var dy = ref.getFullYear()
var dm = ref.getMonth() // represents in numerical value January = 0
var dt = ref.getDate()
var dy = ref.getDay() // represents in numerical value Sunday = 0
var dh = ref.getHours()
var dm = ref.getMinutes()
var ds = ref.getSeconds()
var dms = ref.getMilliSeconds()
```

16/7/23

BOM (Browser Object Model):-

The BOM in js represents various objects & api's (API) provided by the web browsers to interact with the browser windo -w, document & other pages

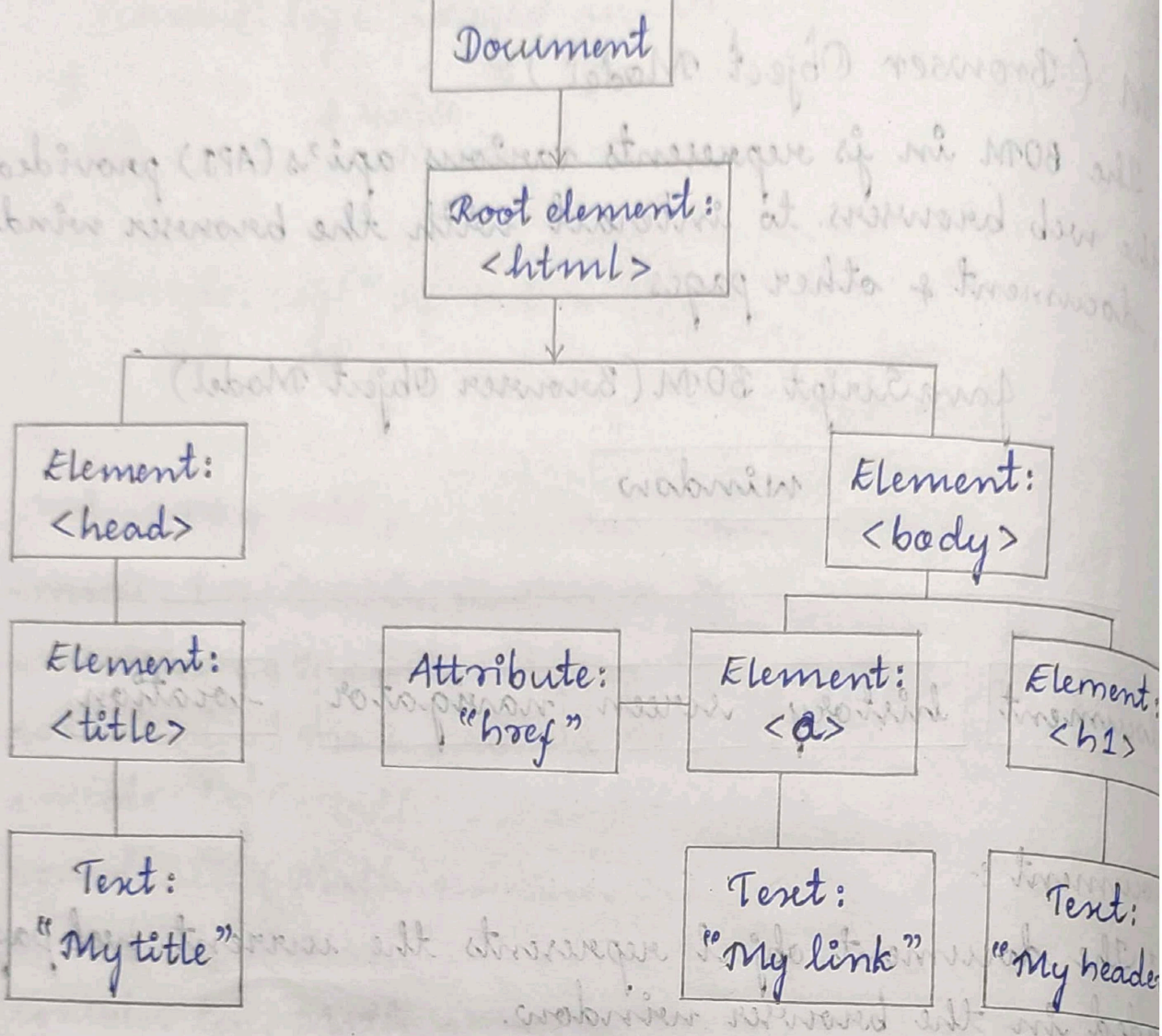
JavaScript BOM (Browser Object Model)



Document :-

* The document object represents the current webpage loaded in the browser window.
* It provides methods & properties for manipulating the contents of document such as accessing & modifing the elements, creating or removing elements & handling events.

DOM (Document Object Model):-



Document Methods :-

1) document.getElementById( ) :-

This method is use to select one html element with the specified id.

```
<div id="div1"> 1st div </div> // html

var div = document.getElementById("div1")
console.log(div);
div.style.backgroundColor = "yellow"
```
// js

2) document.getElementsByClassName( ) :-

This method returns collection of the html elemen in the form Array like objects (html collections) with the specified class name.

```
<main class="main1"> main1 </main>
<main class="main1"> main2 </main>
<main class="main1"> main3 </main>

var main = document.getElementsByClassName("main1")
console.log(main);
console.log(Array.isArray(main));
```

output :-

HTMLCollection(3) [main.main1, main.main1, main.main1]

3) document.getElementsByTagName( ) :-

This method returns collection of html elements in the form of Array like objects with the specified tag name.

```
var ans = document.getElementsByTagName("main");
console.log(ans);
```

output :-

HTMLCollection(3) [main.main1, main.main1, main.main1].

4) document.querySelector( ) :-

This method selects one html element with the specified CSS selector.

```
var ans = document.querySelector(".mainclass");
var ques = document.querySelector("#mainid1");
var tag = document.querySelector("main");
```

5) document.querySelectorAll( ) :-

This method returns collection of html elem in the form of Array like object (node list) for the specified css selector.

```
var ans = document.querySelectorAll("main")
console.log(ans);
```

NOTE :-

* Node list & html collection array are a array lik objects but not a pure array.
* We cannot apply any of the array methods to the html collection & node list.

```
console.log(Array.isArray(ans)); // false
console.log(ans.slice(2)); // error
```

* We can convert impure array (node list & html collection) to a pure array by using Array.from(

```
var convertedArr = Array.from(ans);
console.log(convertedArr.slice(2));
```

18|7|23

1) Create Element( ) :-

This method is use to create html elements throu javascript.

```
var mydiv = document.createElement("div");
```

2) appendChild( ) :-

This method is use to append a single element has a child of another element.

* It takes single arguement which is to be appended has a child.

```
document.body.appendChild("mydiv");
var subdiv = document.createElement("div");
    mydiv.appendChild(subdiv);
var para1 = document.createElement("p");
var para2 = document.createElement("p");
    subdiv.append(para1, para2);
```

) append() :-

This method was introduced in 2019.

This method allows multiple html elements to the parent element in a single function call.

Content can be written in 3 ways

) innerHTML() :-

⇒ These are the properties use to manipulate or retreive the text content of an html element.

* It is a property that represents the html content with in a element including any nested element.

* It allows to set or retrieve the html structure & content of an element.

```
para1.innerHTML = `<table>
    <tr>
      <td><h1>Hi</h1></td>
      <td><h2>Hello</h2></td>
    <tr>
    <table>`
```

2) innerText :-

It is a property that represents the visible text content of an element excluding an html tags.

```
para2.innerText = `<div>
<main>
<h2> hello </h2>
</main>
</div>`
```

output :-

```
<div>
<main>
<h2> hello </h2>
</main>
</div>
```

retrieving

```
var aside = document.querySelector("fig")
console.log(aside.innerText);
```

output :-

hello.

3) textContent :-

```
para3.textContent = `<div>
<main>
<h2> Text Content </h2>
</main>
</div>`
```

19/7/23

Events :-

* In javascript events refers to action or occurance that happen with in a web page or web appl".

Some of the commonly used event types are

1) Mouse Events
2) KeyBoard Events
3) Load Form Events

* Events in javascript allows to create interactive & responsive applications.

addEventListener() :-

* This method is use to attach an event listener to an element.

* It allows to specify a function (event handler) that will be executed when ever the event trigger.

```
addEventListener("click", function()
{
  console.log("I am event handler function");
})
```

html

```
<button> click me <button>
```

js

```
var btn = document.querySelector("button")
btn.addEventListener("click", function()
{
  console.log("I am event handler function");
})
```

html

```
<button onclick = fes()> click me </button>
```

21/7/22

Bubbling and Capturing.

In the context of event handling in java scri bubbling, capturing and target are concepts related to the ~~prog~~ propogation of the event in DOM tree.

They describe the order in which the event han -ers are executed / triggered when an event occurs on an element.

Event Bubbling :-

Event Bubbling is the default behaviour in wh an event is firsthandled by the innermost elemen that triggered the event then propogated to its paren element in the DOM tree.

Event Capturing :-

Event Capturing is the opposite of the Bubbling it allows to capture the events and handle the events at the higher level elements before they reach the target or any child element.

html

```
<div>
  Grand Parent
  <main>
    parent
    <aside>
      child
    </aside>
  </main>
</div>
```

```
var gp = document.querySelector("div")
var p = document.querySelector("main")
var c = document.querySelector("aside")

gp.addEventListener("click", (e) => {
    gp.style.backgroundColor = "red"
    console.log("I am a great parent");
    // e.stopPropogation();
}, true) -> capturing
(or)
}, false) -> bubbling

p.addEventListener("click", (e) => {
    p.style.backgroundColor = "green"
    console.log("I am a parent");
    e.stopPropogation();
}, true)

c.addEventListener("click", (e) => {
    c.style.backgroundColor = "yellow"
    console.log("I am a child");
}, true)
```

Event Target :-

Event Target is a element on which the event was originally triggered.

28/7/23

Promise( ) :-

Promise is an object which is used for asynchronus operation,

* It represents the eventually completion or failure of an asynchronus operation and allows to handle the result when available

* Promise has or it can be of any of the three state

1) Pending :-

It is the initial state when the asynchronus is still not on-going.

2) Fullfilled :-

The state when the asynchronus operation has completed successfully & the result is available.

3) Rejected :-

The state when the asynchronus operation [an encount] come across with any error or failure & error message will available.

* Once a promise is created we can handle a fulfillment or rejection using then catch methods.

then( ) :-

The then( ) method takes a call back function that will be executed when the promise is fulfilled & it recieves the resolved value as an arguement.

catch( ) :-

The catch() method takes a call back function that will be executed when the promise is rejected & it recieves error value as an arguement.

NOTE :-

then() method will allows to perform sequential asynchronus operation.

1/8/23

Storage :-

In javascript storage object is use to store or retreive data in client side.

We have 2 types

(1) Local Storage

(2) Session Storage

(1) Local Storage :-

* It allows to store the data in the form of key value phase.
* The data stored in local storage will be available even after the browsed is closed & reopened.

1) localStorage.setItem("name", "raju")
2) console.log(localStorage.getItem("name"))
3) localStorage.removeItem("name")
4) localStorage.clear( )

2) Session Storage :-

It is also similar to the local storage object but the data stored in the session storage will not be available once the session / browser has been closed & reopened.

```
sessionStorage.setItem("name", "raju")
console.log(sessionStorage.getItem("name"))
sessionStorage.removeItem("name");
sessionStorage.clear();
```

JSON :-

"Java Script Object Notation."

JSON is a popular data format which is used for storing & exchanging the data.

* It is use to transfer data b/w a server & a web application or to store data locally.

JSON.parse()

This method converts JSON object to the java - script object.

JSON.stringify()

This method converts java script object to the JSON object.

ex:-

```
var student = {
    name1 : "raju",
     id : 123,
     0: "zero value"
};
```

```
var jsonObj = JSON.stringfy(student);
console.log(jsonObj);
var jsObject = JSON.parse(jsonobj);
console.log(jsobject);
```

output :-

{"0" : "zero value", "name1" : "raju" , "id" : 123}

{0 : "zero value", name1 : "raju", id : 123}

18/23

Fetch( ) :-

* Fetch( ) method is use to retrieve data from the server.
* It returns a promise that resolves the response object.
* You can handle the response by chaining the .than( ) to the fetch( ).

Syntax :-

```
var ans = fetch("url").then((res) =>
    {
        return res.json( )
    }).then((finvalue) => {
        console.log(finvalue);
    }
console.log(ans);
```

async and await :-

* In js async & await are keywords used to hand asynchronous operation and to make asynchronous cod appear synchronous & readable..
* It was introduced in 2017 ES8.

```
async function foo( ) {
    var response = await fetch("url");
    var finvalue = await response.json( )
        console.log(finvalue);
}
foo( )
```

settimeout( ) and setinterval( ) :-

There are the functions used for executing code after the specified delay or regular interval.

settimeout( ) :-

* This function executes the callback function after the specified delay in milli seconds.
* It takes two parameter.
* A callback function and a time in milli seconds

```
setTimeout(( ) => {
        console.log("I will execute");
}, 2000 after 2 sec");
```

```
setTimeout(() => {
    console.log("I will execute after 2 sec");
}, 2000)
```

setInterval():-

This function is use to repeatedly execute a function at fixed interval.

```
setInterval(() => {
    console.log("I will execute after 2 sec");
}, 2000);
```