JavaScript (Back-End)

1. **What is JavaScript ?**

   ▼ **Ans**

   - **JavaScript is the High level, Object Oriented, Functional Programing Language.**
   - **JavaScript will provide the Brain and Mussels for the Websites, Because JavaScript allows to interact with HTML, based on user given inputs. This makes HTML as a dynamic.**
   - **JavaScript is User Friendly Programing Language.**
   - **JavaScript is an Interpreted Programing Language. (Interpreted means line by line Execution)**
   - **JavaScript is a Scripting Programing Language, that means where instruction are written for Run Time Environmental. It does not requires Compilation step and rather it interpreted line by line execution at the Run Time. Scripting Languages are mainly used to develop Dynamic Web Application. Scripting Languages are written inside <script> Tag.**
   - **JavaScript is a Functional Programing Language. that means It will provide the Functionalities like Onclick, Onchange, Onsubmit etc. this will makes HTML as a dynamic.**
   - **React JS is a Library of JavaScript which was introduced by organization called Facebook in the year of 2013.**
   - **Angular JS is a Frameworks of JavaScript which was introduced by organization called Google in the year of 2007.**
   - **Node JS is the Server Side language of JavaScript, It is not Framework or Library.**

   ▼ **Note**

   - We should write JavaScript code <script> tag.
   - We should write <script> tag inside <body>.

2. **Why we need JavaScript ?**

   ▼ **Ans**

   - **JavaScript is needed to create Dynamic Web Page.**
   - **JavaScript is needed to add the Functionality to the Web page.**

▼ What Kind of Functionality we can add to Web Page ?

▼ Window Event Attributes

- onafterprint
- onbeforeprint
- onbeforeunload
- onerror
- onhashchange
- onload
- onmessage
- onoffline
- ononline
- onpagehide
- onpageshow
- onpopstate
- onresize
- onstorage
- onunload

▼ Form Events

- onblur
- onchange
- oncontextmenu
- onfocus
- oninput
- oninvalid
- onreset
- onsearch
- onselect
- onsubmit

▼ Keyboard Events

- onkeydown

- onkeypress
- onkeyup

▼ **Mouse Events**

- onclick
- ondblclick
- onmousedown
- onmousemove
- onmouseout
  onmouseover
- onmouseup
- onmousewheel
- onwheel

▼ **Drag Events**

▼ **Clipboard Events**

▼ **Media Events**

▼ **Misc Events**

▼ **What do you mean by Static Web Page?**

- **In the Static Web Page data will be static. that means data will be remains same for longer time.**

▼ **What do you mean by Dynamic Web Page?**

- **In the Dynamic Web Page data will be dynamic. that means data will be keep on changing from time to time.**

2. **Is JS, Loosly typed (dynamically type) or strictly type language ?**

▼ **Ans**

- **JS is Loosly Typed Language.**

2. **What is the Advantage of JavaScript ?**

▼ **Ans**

1. **JavaScript can add new HTML Elements and Attributes in the Web Page.**
2. **JavaScript can Change and Remove all the existing HTML Elements and Attributes in the Web Page.**
3. **JavaScript can Change all the CSS Styles in the Web Page.**
4. **JavaScript can add new HTML Events in the Web Page.**

5. **JavaScript can React to all the existing HTML Events in the Web Page.**

2. **Advantages of JavaScript to user ?**

   ▼ **Ans**

   1. **Easier to start with**
   2. **Wide range of usage**
   3. **Big Community Support**
   4. **It is the future**

4. **Software to download for JavaScript ?**

   ▼ **Ans**

   - **Visual Studio, Atom, Sublime text 3**
   - **Extensions 1. Live server 2. node.js**

3. **How JavaScript works?**

   ▼ **Ans**

   - **JavaScript code Runs in Browser, Because the JavaScript Engines are in Browser.**
     - **For chrome - V8 engine**
     - **For Firefox - Spidel Monkey**

4. **What V8 engine?**

   ▼ **Ans**

   - **V8 is a High-Performance, Open-Source, Web Assembly JavaScript Engine used by Google Chrome and Node.js**
   - **3 steps involved in processing the code:**
     1. **Parsing the code.**
     2. **Compiling the code.**
     3. **Executing the code.**

5. **How many printing statements are there in JavaScript ?**

   ▼ **Ans**

   In JavaScript, We have two printing statements

   1. **document.write("");**
   2. **console.log("");**

6. **Difference between document.write() and console.log() ?**

   ▼ **Ans**

document.write()

- **document.write() is a composed printing statement, Where document is an Object and write() is function, Which is used to print the output on the Web Browser as Client side.**

console.log()

- **console.log() is a composed printing statement, Where console is an Object and log() is function, Which is used to print the output on the console window as Developer side.**

▼ **Program**

```
<script>
      document.write("JS Class");
      console.log("Don't be late");
 </script>
```

7. **What is Keyword ?**

▼ **Ans**

- **Keywords are Reserved words which has predefined meaning to it. All keywords are in lower case. There is no fixed number of keyword in JS.**

  (var , let , const , constructor , function , with , class , if , for , while , else , switch , ease , break , this , ...etc)

8. **What is Variable ?**

▼ **Ans**

- **Variable is nothing but a container which stores data, It is also known as data holder.**

In JavaScript, Variable can be declared in 4 ways

1. **varName = Data**
2. **var varName = Data**
3. **let varName = Data**
4. **const varName = Data**

▼ **Program**

```
<script>
        x=100;
        document.write(x+"<br>")
        console.log(x)
        var x1=true;
        document.write(x1+"<br>")
        console.log(x1)
```

```
        let x2="JS Class";
        const pi=3.14;

        document.write(x2+"<br>")
        console.log(x2)

        document.write(pi+"<br>")
        console.log(pi)
    </script>
------------OUTPUT------------
100
true
JS Class
3.14
```

9. **What is the difference between var, let and const ?**

▼ **Ans**

▼ `var`

- `var` keyword is used **to declare a variable in JavaScript.**

▼ `var` Variable can be Re-declared.

▼ `var` Variable can be Re-declared.

```
<script>
        var a = 10
        var a = 8
        console.log(a)
    </script>
---------------OUTPUT-----------------------
8
```

▼ `var` Variable can be Re-declared in anywhere in the program

```
<script>
            var x=100;
            {
                var x=101;
            }
            function manu()
            {
                {
                    var x=105;
                }
               console.log(x)
            }
            manu()
            console.log(x);
    </script>
-----------------OUTPUT-----------------------------
105
101
```

▼ `var` Variable does not have Block Scope

- `var` Variable declared inside a{ } block can be accessed from outside the block.

```
<script>
        let a = 10;
        function f()
        {
            if (true)
            {
                var b = 9
                console.log(b);//prints 9
            }

            console.log(b);//print 9
        }
        f()
        console.log(a)
    </script>
--------------OUTPUT------------------------------
9
9
10
```

▼ If `var` Variable is declared in globally, then it be can access inside the function or outside the function.

```
<script>
        var a = 10
            function f()
            {
                console.log(a)
            }
        f();
        console.log(a);
    </script>
-----------OUTPUT-----------------
10
10
```

▼ `var` variable must be Declared before use.

```
<script>
    console.log(a);
    var a = 10;
<script>
-----------OUTPUT-----------------
undefined
```

▼ `var` variable can be declared without initialization.

```
<script>
            var x;
            x=100;
            console.log(x)
```

```
    </script>
----------OUTPUT------------
100
```

▼ `var` variable Dis-advantage

▼ Redeclaring `var` variable can impose problems.

```
<script>
        var x=10;
            if(true)
            {
                var x=12;
            }
        console.log(x);  //12
    </script>
--------------OUTPUT----------------------------
12
```

▼ `let` variable can solve this problem. Because it has block scope

```
<script>
        let x=10;
            if(true)
            {
                let x=12;
            }
        console.log(x);
    </script>
------------OUTPUT------------------
10
```

▼ **`let`**

- `let` is a keyword used to declare variables that are block scoped and we can modify its value but we can redeclare it.

▼ `let` variable cannot be Re-declared.

▼ `let` variable cannot be Re-declared.

```
<script>
        let a = 10
        let a = 10
        console.log(a);
    </script>
------OUTPUT-----------
SyntaxError: Identifier 'a' has already been declared

--------By using var It is possible---------------
<script>
        var a = 10
        var a = 10
```

```
        console.log(a);
    </script>
------OUTPUT-----------
10
```

▼ `let` variable cannot be Re-declared but can be update it.

```
<script>
        let a = 10
            a = 11
        console.log(a);
    </script>
--------------OUTPUT--------------------------------
11
```

▼ `let` variable Re-declaring in the same block is not allowed, but in another block is allowed.

```
--Re-declaring in the same block is not allowed----
<script>
          let x = 2;   //Allowed
          let x = 3;   // Not allowed
          {
             let x = 2;   // Allowed
             let x = 3;   // Not allowed
          }
          {
              let x = 2;   // Allowed
              var x = 3;   // Not allowed
           }
    </script>
--Re-declaring in the another block is allowed----
<script>
             let x = 2;   //Allowed
            {
            let x = 3;   // Allowed
            }

            {
            let x = 4;    // Allowed
            }
            console.log(x) //2
    </script>
-------OUTPUT---------
2
--Re-declaring in the another block is allowed(2)-----
<script>
        let a = 10
        if (true)
         {
            let a=9
            console.log(a) // It prints 9
          }
        console.log(a) // It prints 10
        </script>
-------OUTPUT-------
9
10
```

▼ `let` variable must be Declared before use.

```
<script>
    console.log(a);
    let a = 10;
</script>
---------------OUTPUT-----------------------------
Uncaught ReferenceError: Cannot access 'a' before initialization
```

▼ `let` variable have Block Scope.

- Variables declared inside a { } block cannot be accessed from outside the block:

```
<script>
        let a = 10;
        function f()
        {
            if (true)
            {
                let b = 9
                console.log(b);//prints 9
            }

            console.log(b);//error as it defined in if block
        }
        f()
        console.log(a)
    </script>
------------------OUTPUT-----------------------------
9
ReferenceError: b is not defined
```

▼ `let` variable can be declared without initialization.

```
<script>
            let x;
            x=100;
            console.log(x)
    </script>
----------OUTPUT------------
100
```

▼ `let` variable Advantage over `var` variable

▼ Redeclaring `var` variable can impose problems.

```
<script>
        var x=10;
            if(true)
            {
                var x=12;
            }
```

```
        console.log(x);  //12
    </script>
---------------OUTPUT----------------------------
12
```

▼ `let` variable can solve this problem. Because block scope Redeclaring a variable inside a block will not redeclare the variable outside the block.

```
<script>
        let x=10;
            if(true)
            {
                let x=12;
            }
        console.log(x);
    </script>
-------------OUTPUT-------------------
10
```

▼ When to use JavaScript `let` Keyword ?

- If we want value of the variable can change, we make use `let` keyeord

▼ `const`

- `const` is a keyword used to declare constants variables that are block scoped, much like variable declared using the let keyword.

▼ `const` variable cannot be Re-declared.

▼ `const` variable cannot be redeclared but `var` can be redeclared

```
------------Using var it is posible-----------------------
<script>
    var x = 2;
    var x = 3;
    x = 4;
    console.log(x)
    </script>
------OUTPUT------
4

(but)

------------Using const it is not posible-------------------
<script>
    const x = 2;
    const x = 3;
    x = 4;
    console.log(x)
    </script>
------------OUTPUT---------------
Uncaught SyntaxError: Identifier 'x' has already been declared
```

- Redeclaring an existing `var` or `let` variable to `const`, in the same scope, is not allowed

```
<script>
        var x = 2;
        const x = 2;

        {
            let x = 2;
            const x = 2;
        }

        {
            const x = 2;
            const x = 2;
        }
    console.log(x)
    </script>
```

- Redeclaring a variable with `const`,in another scope, or in another block, is allowed:

```
<script>
        const x = 2;       // Allowed

        {
            const x = 3;   // Allowed
        }

        {
            const x = 4;   // Allowed
        }
        console.log(x);
    </script>
--------------OUTPUT----------------
4
```

- `const` variable cannot be Re-assigned.
  - `const` variable cannot be Re-Assigned

```
<script>
        const PI = 3.141592653589793;
        PI = 3.14;
        PI = PI + 10;
        console.log(PI)
    </script>
--------------OUTPUT--------------------
Uncaught TypeError: Assignment to constant variable.
```

  - `const` variable must be assigned a value when they are declared

```
<script>
       const PI;
        PI = 3.14159265359;
        console.log(PI);
    </script>
---------------OUTPUT-------------------------
Uncaught SyntaxError: Missing initializer in const declaration


<script>
       const PI = 3.14159265359;
        console.log(PI);
    </script>
-----------------OUTPUT--------------------------
3.14159265359
```

▼ `const` variable have Block Scope.

- Variables declared inside a { } block cannot be accessed from outside the block:

```
<script>
        const a = 10;
        function f()
        {
            if (true)
            {
                const b = 9
                console.log(b); //prints 9
            }

            console.log(b); //error as it defined in if block
        }
        f()
        console.log(a)
    </script>
------------------OUTPUT-----------------------------
9
ReferenceError: b is not defined
```

▼ `const` variable cannot be declared without initialization.

```
<script>
            const x;
            x=10;
            console.log(x)
    </script>
----------OUTPUT------------
Uncaught SyntaxError: Missing initializer in const declaration
```

▼ When to use JavaScript `const` Keyword?

- If we want value of the variable cannot be changed or constant value, then we make use `const` keyeord

▼ Use `const` variable whenever we are declaring :-

1. for new Array
2. for new Object
3. for new Function
4. for new RegExp

▼ Note

- It does not define a constant value. It defines a constant reference to a value. Because

  ▼ We can change the elements of a `const` Array, But you can NOT reassign the `const` Array

```
<script>
    const cars = ["Saab", "Volvo", "BMW"];

        cars[0] = "Toyota";
        cars.push("Audi");
        console.log(cars)
    </script>
-----------------OUTPUT----------------------
['Toyota', 'Volvo', 'BMW', 'Audi']
0: "Toyota"
1: "Volvo"
2: "BMW"
3: "Audi"
-------------------------------------------------
<script>
    const cars = ["Saab", "Volvo", "BMW"];

    cars = ["Toyota", "Volvo", "Audi"];
    </script>
------------OUTPUT--------------------
Uncaught TypeError: Assignment to constant variable
```

  ▼ You can change the properties of a `const` Object, But you can NOT reassign the `const` Object:

```
<script>
    const car = {
            type:"Fiat",
            model:"500",
            color:"white"
                };
    car.color = "red";
    car.owner = "Johnson";
    console.log(car);
    </script>

-------------------------------------------------

<script>
    const car = {type:"Fiat", model:"500", color:"white"};
```

```
    car = {type:"Volvo", model:"EX60", color:"red"};
    console.log(car);
    </script>
--------------OUTPUT-----------------
Uncaught TypeError: Assignment to constant variable.
```

`var`	`let`	`const`
1). Scope of a *var* variable is Functional scope.	**1). Scope of a *let* variable is Block scope**	**1). Scope of a *const* variable is Block scope**
2). *var* variable can be Updated and Re-declared into the scope	**2). *let* variable can be updated but it cannot be Re-declared into the scope**	**3). *const* variable cannot be updated and Re-declared into the scope.**
3). *var* variable can be declared without initialization,	**3). let variable can be declared without initialization**	**3). *const* variable cannot be declared without initialization**
4). *var* variable can be accessed without initialization and its default value is “undefined”.	4). let variable cannot be accessed without initialization otherwise it will give ‘reference Error’.	4). *var* variable cannot be accessed without initialization, as it cannot be declared without initialization.
5). hoisting done, with initializing as ‘default’ value.	5). Hoisting is done , but not initialized (this is the reason for the error when we access the let variable before declaration/initialization.	5). Hoisting is done, but not initialized (this is the reason for error when we access the const variable before declaration/initialization.

11. **What is if ?**

    ▼ Ans

    - **Whenever the condition is true if will Execute, Whenever the condition is false if will not Execute.**

12. **What is the difference between == and === ?**

    ▼ Ans

Operator	Data	Data-Type
==	yes	no
===	yes	yes

    ▼ Example Program for ==

```
<script>
       if(100=="100")
```

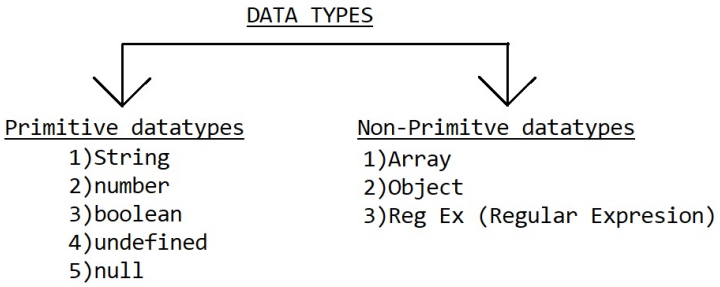
```
        {
         document.write("This is Java")
        }
        else
        {
         document.write("This is JavaScript")
        }
    </script>
----------OUTPUT----------
This is Java
```

▼ **Example Program for ===**

```
<script>
        if(100==="100")
        {
            document.write("This is Java")
        }
        else
        {
            document.write("This is JavaScript")
        }
    </script>
------------OUTPUT-------------
This is JavaScript
```

13. **What is Data-Types ?**

▼ **Ans**



- **Data-Types describes the what kind of data we are going store in memory allocation.**
- **We have 2 types Data-Types in JS**

▼ **Primitive Data-Type**

- **In JavaScript, We have 5 Primitive Data-Types are there, Which can store only one value or data.**

▼ **String**

- **A bunch of character or set of character forms a String.**
- **We can represent String in 3 way**

1. **Using double quotes (“”)**
2. **Using single quotes (’)**
3. **Backticks (``) (present above tab bottom) (Backticks are also called multi line String)**

▼ **Example Program 1**

```
<script>
        var firstname="dashwath"
        var lastname='kotien'
        var fullname=`dashwath kotian`
        document.write(firstname+"<br>")
        document.write(typeof(firstname)+"<br>")

        document.write(lastname+"<br>")
        document.write(typeof(fullname)+"<br>")
    </script>
-------------OUTPUT------------------
dashwath
string
kotien
string
```

▼ **Example Program 2**

```
console.log(2+1);
console.log("2"+1);
console.log(2-1);
console.log("2"-1);
console.log("2"-"2");
-------------OUTPUT--------------
3
"21"
1
1
0
```

▼ **number**

- **Whenever we want to store decimal and non-decimal numeric, we make use of number datatype.**

▼ **Example Program**

```
<script>
        var x=100;
        var x1=34.67;
        var bankbalance=399456586996466484864199643547826425429282n

        document.write(x+"<br>")
        document.write(typeof(x)+"<br>")

        document.write(x1+"<br>")
```

```
        document.write(typeof(x1)+"<br>")

        document.write(bankbalance+"<br>")
        document.write(typeof(bankbalance)+"<br>")
     </script>
------------OUTPUT------------
100
number
34.67
number
399456586996466484864199643547826425429282
bigint
```

▼ boolean

- **Whenever we have decision making scenario boolean datatype which is capable of giving two values. (true or false)**

▼ Example Program

```
<script>
       var x=true
       document.write(x+"<br>")
       document.write(typeof(x)+"<br>")
     </script>
---------OUTPUT-------------
true
boolean
```

▼ undefined

- **undefined is the default value of any variable utill unless initialization happens.**

▼ Example Program

```
<script>
      var x;
      document.write(x+"<br>")
      document.write(typeof(x)+"<br>")
 </script>
-----------OUTPUT-----------
undefined
undefined
```

▼ null

- **null refers nothing or empty. Whenever we want to make any container as a empty container, We make use of null datatype, we have to pass null as data.**

▼ Example Program

```
<script>
        var x=789;
        var x=null;
        document.write(x+"<br>")
        document.write(typeof(x)+"<br>")
   </script>
-------------OUTPUT------------
null
object
```

▼ **Non-Primitive Data-Type**

- **Non-Primitive Data-Types which can store n number of value or data. Example: Array, Object, Functions.**

▼ **3 Special values**

- **In JS, We have 3 special values**
  1. **Infinity**
  2. **-Infinity**
  3. **NaN (Not-A-Number)**

▼ **Example Program**

```
<script>
       document.write(100/0+"<br>")
       document.write(-100/0+"<br>")
       document.write("Java"/100+"<br>")
     </script>
----------OUTPUT--------------
Infinity
-Infinity
NaN
```

14. **What is copy past approach ?**

▼ **Ans**

▼ **In copy pasting approach we are facing 3 major problem**

1. Code looks lengthier.
2. Code redundancy or code duplication.
3. Modification is done in original code will not effect in the pasted code.

- **To overcome with three major drawbacks, we are going for the concept called functions.**

▼ **Example Program for Copy past**

```
<script>
       document.write("========RCB==========="+"<br>")
       document.write("Virat"+"<br>")
       document.write("Abd"+"<br>")
       document.write("Dk"+"<br>")
       document.write("======CSK========="+"<br>")
       document.write("MSD"+"<br>")
       document.write("Jadeja"+"<br>")
       document.write("Ambati"+"<br>")

       document.write("======CSK========="+"<br>")
       document.write("MSD"+"<br>")
       document.write("Jadeja"+"<br>")
       document.write("Ambati"+"<br>")
       document.write("========RCB==========="+"<br>")
       document.write("Virat"+"<br>")
       document.write("Abd"+"<br>")
       document.write("Dk"+"<br>")
     </script>
----------------OUTPUT------------
========RCB==========
Virat
Abd
Dk
======CSK=========
MSD
Jadeja
Ambati
======CSK=========
MSD
Jadeja
Ambati
========RCB==========
Virat
Abd
Dk
```

20. **Write a JavaScript program Reverse given string ?**

▼ **Ans**

```
-----------------Program 1--------------------------------------------
<script>
      var str="bangalore"
      console.log(str)

      var arraystr=str.split("")
      console.log(arraystr)

      var rev=arraystr.reverse();
      console.log(rev)

      var result=rev.join("")
      console.log(result)

     </script>
----console OUTPUT--------
   ['b', 'a', 'n', 'g', 'a', 'l', 'o', 'r', 'e']
   ['e', 'r', 'o', 'l', 'a', 'g', 'n', 'a', 'b']
  erolagnab
```

```
-----------------Program 2-------------------------------------------
<script>
      console.log("bangolre".split("").reverse().join(""))
     </script>
----console OUTPUT-----------
  erolagnab
```

21. **What is Math.random() and Math.floor() ?**

    ▼ **Ans**

    - **Math.random() function returns the floating-point random number between 0 to 1.**
    - **Math.floor() function is used to round off the number which is passed as a parameter and also it will remove floating value.**

21. **How to generate 4digit number in JS ?**

    ▼ **Ans**

    - Math.random(); —> To get random number
    - To get 4digit, We have to multiply with 1000. In this case there is a chance to get 1,2,3,4, digit, but we want 4digit only. so use Math.random()*(H.N-L.N)+L.N for digit. E.x = Math.random()*(9999-1000)+1000; it will generate 4 digit value.
    - If we want to remove the floating value pass this to Math.floor();

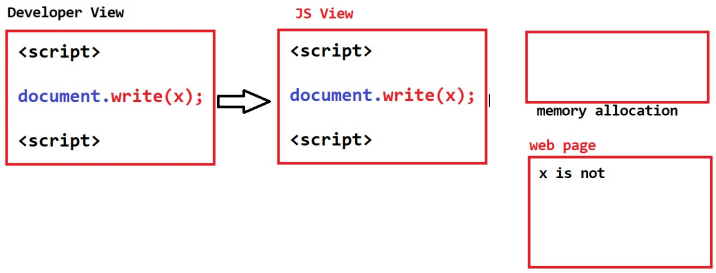
```
var x=Math.random();  //0 to 1 -->floating value
console.log(x);
console.log(x*10000);

var result=Math.random()*(9999-1000)+1000;   //To get 4digit value
console.log(result)

var otp=Math.floor(result)   //To floating value
console.log(otp)
------------------OUTPUT--------------
0.6164969312269211
6164.969312269211
3293.286956372209
3293
```

23. **What is Variable Hoisting ? Explain Variable Hoisting with an Example ?**

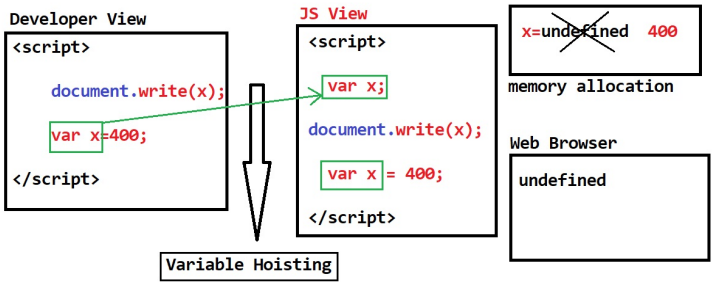
    ▼ **Ans**

    - **Moving variable declaration at the top of original code or native code this process is called Variable Hoisting. Variable Hoisting will take place only when we declare a variable with “var” keyword.**

    ▼ **Scenario 1**



▼ Scenario 2



▼ Example 1

```
<script>
var x=100;
var r="hi"
document.write(x+"<br>")
var x=400;
document.write(r+"<br>")
var x=400;
document.write(r+"<br>")
document.write(d+"<br>")
document.write(c+"<br>")
var r=89;
var c=true;
var s="variable"
document.write(x+"<br>")
var d=765;
</script>
----------------OUTPUT---------------
100
hi
hi
undefined
undefined
400
```

24. **What is http and https ?**

▼ Ans

- **http (Hypertext Transfer Protocol) —> open source**
- **https (Hypertext Transfer Protocol Secure) —> secured**

25. Scope

    ▼ **Ans**

    - **Scope is nothing but which described boundary or visibility. In JS we have 2 Scope.**

        ▼ **Global Variable**

        - **A Variable which is declared outside the Function scope, But inside the Script Tag is called Global Variable. The visibility of Global Variable is any where in the Script Tag.**

        ▼ Global Scope

        ▼ Local Scope

26.

27.

20. What is difference between Rest and Normal parameter ?

    ▼ Ans

15. **What is Functions ?**

    ▼ **Ans**

    - **In order to Execute some set of codes again and again according to user requirement. We use Functions. Functions can be invoked using function name followed by function notation. Without calling function, function will not going to be executed.**
    - **We can declare a function before function call as well as after function call.**
    - **Syntax**

```
function functionName()
{
   //statement
}
```

    ▼ **Example Program 1**

```
<script>
        function RCBTeam()
       {
         document.write("========RCB==========="+"<br>")
         document.write("Virat"+"<br>")
         document.write("Abd"+"<br>")
         document.write("Dk"+"<br>")
        }
        function CSKTeam()
        {
         document.write("======CSK========="+"<br>")
```

```
         document.write("MSD"+"<br>")
         document.write("Jadeja"+"<br>")
         document.write("Ambati"+"<br>")
        }

        RCBTeam()  //function call
        CSKTeam()  //function call
        CSKTeam()  //function call
        RCBTeam()  //function call
    </script>
-------------OUTPUT----------------------------------
========RCB==========
Virat
Abd
Dk
======CSK=========
MSD
Jadeja
Ambati
======CSK=========
MSD
Jadeja
Ambati
========RCB==========
Virat
Abd
Dk
```

▼ Example Program 2

```
<script>
        function Student()
        {
            document.write("Student name is raj"+"<br>")
            document.write("Raj is from Bangalore"+"<br>")
            document.write("<br>")
        }
        Student()//function call
        Student()//function call
        Student()//function call
        Student()//function call
        Student()//function call
    </script>
-------------------OUTPUT-----------------------------
Student name is raj
Raj is from Bangalore

Student name is raj
Raj is from Bangalore

Student name is raj
Raj is from Bangalore

Student name is raj
Raj is from Bangalore

Student name is raj
Raj is from Bangalore
```

- From the above example we note that for each function call, we are getting same output hardcoded.
- **To overcome with this problem, we are going for concept called parameterized function.**

16. **What are the types of Function ?**

▼ **Ans**

▼ **Parameterized Function**

- **Declaring a parameter in the function definition is called Parametrized Function.**
  - **Parameter is container which holds the Argument.**
  - **Argument is the data which is passed in function call.**
- **Syntax**

```
function funcationName(par1,par2)
{
    //statement
}
```

▼ **Example 1**

```
<script>
        function Student(name,place)
        {
            document.write("Student name is"+name+"<br>")
            document.write(name+"is from"+place+"<br>")
            document.write("<br>")
        }
        Student("Raj","Bangalore") //function call
        Student("Kumar","Maglore")  //function call
        Student("Raj Kumar","Andra")  //function call
        Student("Puneeth raj Kumar","Karnataka") //function call
    </script>
------------------OUTPUT----------------------
Student name isRaj
Rajis fromBangalore

Student name isKumar
Kumaris fromMaglore

Student name isRaj Kumar
Raj Kumaris fromAndra

Student name isPuneeth raj Kumar
Puneeth raj Kumaris fromKarnataka
```

▼ **Example 2**

```
<script>
        function Student(A,B,C)
        {
            document.write("A value is "+A+"<br>")
            document.write("B value is "+B+"<br>")
            document.write("C value is "+C+"<br>")
            document.write("<br>")
        }
        Student(100,200,300)    //function call
        Student(23,true)       //function call
        Student("JavaScript") //function call
        Student()            //function call
     </script>
--------------------OUTPUT----------------
A value is 100
B value is 200
C value is 300

A value is 23
B value is true
C value is undefined

A value is JavaScript
B value is undefined
C value is undefined

A value is undefined
B value is undefined
C value is undefined
```

- From the above Example 2, We note that passed parameter need not be same as passed arguments.
- In parameterized function datatype checking will not happen.

▼ Example 3

```
<script>
        function display(A,B,C)
        {
            document.write("A value is "+A+"<br>")
            document.write("B value is "+B+"<br>")
            document.write("C value is "+C+"<br>")
            document.write("<br>")
        }
        display(100,200,300,78,90,87,566)  //function call
       //[argument Object]= 100,200,300,78,90,87,566
     </script>
-----------OUTPUT-------------
A value is 100
B value is 200
C value is 300
```

▼ Example 4

```
<script>
        function display()
        {
            document.write(arguments[0]+"<br>")
            document.write(arguments[2]+"<br>")
            document.write(arguments[770]+"<br>")
            document.write(arguments+"<br>")
        }
        display(100,200,300,78,90,87,566)  //function call
     </script>
-----------------output-----------------
100
300
undefined
[object Arguments]
```

- **From the above example, We note that data are directly fetch from argument object.**
- **The data which is stored in function call. First, It will move to argument object, from argument object it will check for memory allocation to store the data which present inside argument object.**
- In argument object, data are stored in form of array. using index value we can fetch the data directly from the argument object

▼ Default Parameterized Function

- **Default Parameter is used to set the default values for function parameters is called Default Parameterized Function.**
- **Syntax**

```
function functionName(para1,para2,,,,,,,para N=vale)
{
    ///statement
}
```

▼ Example 1 (Default Function)

```
<script>
      function display(A=10,B=12,C=78,D=95)
      {
        document.write("A value is "+A+"<br>")
        document.write("B value is "+B+"<br>")
        document.write("C value is "+C+"<br>")
        document.write("D value is "+D+"<br>")
      }
      display()
     </script>
-----------------OUTPUT-----------------
A value is 10
B value is 12
```

```
C value is 78
D value is 95
```

▼ Example 2 (Default Function with Overridden by Function Arguments)

```
<script>
      function display(A=10,B=12,C=78,D=95)
      {
        document.write("A value is "+A+"<br>")
        document.write("B value is "+B+"<br>")
        document.write("C value is "+C+"<br>")
        document.write("D value is "+D+"<br>")
      }
      display(100,200,300,400)
     </script>
------------------OUTPUT---------------
A value is 100
B value is 200
C value is 300
D value is 400
```

▼ Example 3

```
<script>
      function display(A,B,C,D=95)
      {
        document.write("A value is "+A+"<br>")
        document.write("B value is "+B+"<br>")
        document.write("C value is "+C+"<br>")
        document.write("D value is "+D+"<br>")
        document.write("<br>")
      }
      display(100,200,300,400) //function call
      display(100,200,300) //function call
      display(100,200) //function call
      display(100) //function call
      display() //function call
     </script>
------------------OUTPUT-----------------
A value is 100
B value is 200
C value is 300
D value is 400

A value is 100
B value is 200
C value is 300
D value is 95

A value is 100
B value is 200
C value is undefined
D value is 95

A value is 100
B value is undefined
C value is undefined
D value is 95
```

```
A value is undefined
B value is undefined
C value is undefined
D value is 95
```

▼ Rest Parameter Function

- **Rest parameter is used to make any parameter as an infinite parameter. In Rest parameter, arguments are stored in the form array, By using index value we can fetch the data one by one.**
- **Rest has to be prefixed with 3 dots (…)**
- **We can pass Rest Parameter along with a Normal Parameter, but Rest Parameter must be last Formal Parameter.**
- **Syntax**

```
function functionName(...para)
{
     //Statement
}
functionName(args1,args2,,,,,,,,args n)
```

▼ Example 1 (Rest parameter)

```
<script>
      function display(...para)
      {
        document.write(para)
      }
      display(100,200,"300",78,90,"87",566,true)
     </script>
----------------OUTPUT-----------------
100,200,300,78,90,87,566,true
```

▼ Example 2 (Formal with Rest parameter)

```
<script>
      function display(A,B,...para)
      {
        document.write(A+"<br>")
        document.write(para[1]+"<br>")
        document.write(para)
      }
      display(100,200,300,400,500,600)
     </script>
----------------OUTPUT----------
100
400
300,400,500,600
```

▼ Example 3 (First Rest Parameter After Formal Parameter = get error)

```
<script>
      function display(...para,A)
      {
          //error
          //Rest parameter must be last formal parameter
        document.write(A+"<br>")
        document.write(para)
      }
      display(100,200,300,400,500,600)
     </script>
-----------OUTPUT-----------
error
```

▼ Return Type Function

- **Return Type Function capable of returning the Data, Variable, Expression or Function.**
- **Return Type Function must contains "return" keyword and it must be last executable statement inside the function scope.**
- **Return Type Function Output can be seen in 2 ways :-**
  1. Attach Function Calling statement to a variable and print the variable.
  2. Write Function Calling statement inside any printing statement.

▼ Syntax

```
Synatx 1:     function functionName()
              {
                return data
              }
Synatx 2:   function functionName()
            {
              return variable
            }
Synatx 3:   function functionName()
            {
              return Expreesion
            }
```

▼ Example 1 (Returning Data)

```
<script>
    function display()
    {
        return "Angular-JS"
    }
    var x=display()
    document.write(x)
</script>
```

```
--------------------OUTPUT---------------
Angular-JS
```

▼ **Example 2 (Returning Variable)**

```
<script>
    function display(a,b)
    {
        var z=a+b;
        return z;
    }
    var x=display(12,45)
    document.write(x)
</script>
-----------------------------(or)----------------------------
<script>
    function display(a,b)
    {
        var z=a+b;
        return z;
    }
    document.write(display(12,45))
</script>
------------------------OUTPUT---------------
57
```

▼ **Example 3 (Returning Expression)**

```
<script>
    function display(a,b)
    {

        return 12+89/3*34-89*76;
    }
    var x=display()
    document.write(x)
</script>
----------OUTPUT----------
-5743.333333333333
```

▼ **Anonymous Function [ES-6]**

- **A Function having without Function name is called Anonymous Function. An Anonymous Function has to be stored in a Variable and In order to call the Anonymous Function, We must use of variable name along with Function Notion.**
- **Uses:-**
  1. **To declare the function inside the object.**
  2. **To make Call Back Functions.**
- **Syntax**

```
var varname = function()
              {
                  //statement
              }
varname() //function call
```

▼ Example 1

```
<script>
        var x = function()
                {
                    document.write("java script")
                }

            x()     //function call
</script>
--------------OUTPUT--------------
java script
```

▼ Example 2

```
<script>
        var x = function(subject)
              {
                    document.write(subject+"<br>")
              }

          x("angular js")      //function call
          x("react js")        //function call
</script>
```

▼ Example 3

```
<script>
          var x = function(subject)
                {
                    return subject;
                }

           var result = x("node js")         //function call
           document.write(result)
</script>
------------OUTPUT-----------
node js
```

▼ Example 4 (One Function Returning Another Function)

```
<script>
        function display()
        {
            return function()
```

```
                    {
                      return "Angular JS"
                    }
        }

       var y = display()             //outer function call
       document.write(y())           //inner function call
</script>
----------------OUTPUT--------------
Angular JS
```

▼ Call Back Function

- **Function passing as an argument to another function is called Call Back Function.**
- **Uses:-**

▼ Example 1

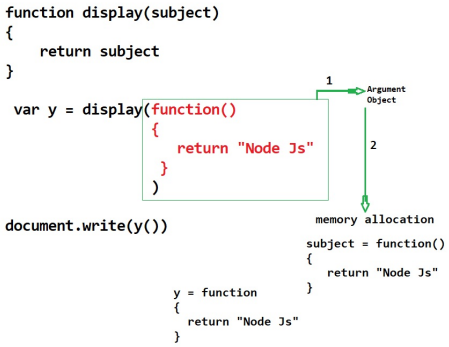
```
<body>
    <script>]

         function display(subject)
          {
            return subject
          }

         var y = display(function()
                       {
                          return "manu"
                       })
    document.write(y()) //Anonymous function call

    </script>
</body>
----------------OUTPUT-------------------------
manu
```

▼ Example 2

```
<script>
     function display(subject)
     {
        return subject
     }

     var y=display(function()
                   {
                    document.write("manu")
                   })
        y() //Anonymous function call
    </script>
--------------------OUTPUT--------------------
manu
```

▼ Example 3 (Prajwal)

```
// let student = [{name:"Rama",id:1},{name:"sita",id:2}]

//     function enrollStudent(newStudent){
//         setTimeout(()=>{
//             student.push(newStudent)
//         },3000)
//     }
//     function getStudent(){
//         console.log(student);
//     }
// enrollStudent({name:"a",id:2})
// getStudent()


    let student = [{name:"Rama",id:1},{name:"sita",id:2}]
    function enrollStudent(newStudent , callback){
        setTimeout(()=>{
            student.push(newStudent)
            callback()
        },3000)
    }
    function getStudent(){
        console.log(student);
    }
enrollStudent({name:"a",id:2} , getStudent)
```



▼ Arrow function [ES-6]

- **Arrow Function is used for call backs to reduce many lines of code to a single line.**
- **Syntax**

```
var varName = () => {statement}
```

▼ Example 1

```
<script>

        var x = () => document.write("arrow function")
```

```
              x()           // function call

</script>
-----------OUTPUT-----------
arrow function
```

▼ Example 2

```
<script>

        var display = (name) => document.write("my name is "+name+"<br>")

            display("Manu")         //function call
            display("Meghraj")        //function call
            display("Anand")        //function call

</script>
---------------OUTPUT------------
my name is Manu
my name is Meghraj
my name is Anand
```

▼ Example 3

```
<script>

  var display = name => document.write("my name is "+name+"<br>")

     display("Manu")             //function call
     display("Meghraj")          //function call
     display("Anand")            //function call

</script>
---------------OUTPUT------------
my name is Manu
my name is Meghraj
my name is Anand
```

▼ Example 4

```
<script>

        var d = (subject) => {return subject}

        document.write(d("React js"))

</script>
-------------OUTOUT----------
React js
```

▼ Example 5

```
<script>

        var d = subject => {return subject}

        document.write(d("React js"))

</script>
------------OUTOUT----------
React js
```

▼ **Example 6**

```
<script>

      var d = subject => subject

      document.write(d("React js"))

</script>
-----------OUTPUT------------
React js
```

▼ **Example 7 (Call-back function using Arrow function)**

```
<script>

        var display = sub => sub

        var x = display(()=>"expess js")    //outer

        document.write(x())               //Inner

</script>
-----------OUTPUT--------
expess js
```

▼ **Rules to write Arrow function**

1. **No need of writing Function Keyword in Arrow Function.**
2. **No need of writing Function Name in Arrow Function.**
3. **We can neglect curly braces, when we have only one printing statement or return statement.**
4. **We can neglect parenthesis, Whenever we have only one parameter.**
5. **We can write Return Type Function without "return" keyword as show in Example 6.**

17. **In JavaScript, Function is Primitive or Non-Primitive ?**

▼ **Ans**

- **It is a Non-Primitive. Because Primitive data-type can hold only one value.**

18. **What is Function Expression ?**

    ▼ **Ans**

    - **If a Function is stored in a variable, We can call it as Function Expression.**

    ▼ **Example 1 (Using Normal Function)**

```
let a = function add(a,b)
        {
            console.log(a+b);
        }
        a(2,3)
--------------OUTPUT-------------
5
```

    ▼ **Example 2 (Using Ananymous Function)**

```
let a = function(a,b)
        {
            console.log(a+b);
        }
        a(2,3)
--------------OUTPUT-------------
5
```

19. **How to call Anonymous Function ?**

    ▼ **Ans**

    - **We can call Anonymous function, By assigning variable to a Anonymous function by using variable name followed by function notation. We can call.**

20. **What is Array ?**

    ▼ **Ans**

    - **Array is non-primitive data type in JavaScript and it is heterogenous in nature. (Heterogenous: It can accept any data type)**

    ▼ **Declare an Array in (3 ways)**

        ▼ **Array Literals**

            ▼ Syntax

```
var VarName = [data1, data2, ......data N];
```

            ▼ Example

```
<script>
        var data = [12,"JS",true,null,23.88];
        document.write(data+"<br>")
        document.write(data[1]+"<br>")
        document.write(data[4]+"<br>")
    </script>
---------------OUTPUT-------------------------------
12,JS,true,,23.88
JS
23.88
```

▼ Using ‘new’ keyword

▼ Syntax

```
  var VarName = new Array();
```

▼ Example 1

```
<script>
        var food = new Array();
        food[0] = "dosa";
        food[1] = "palav";
        food[2] = "chapati";
        food[3] = "vada";

        console.log(food)
        console.log(food[3])
        console.log(food[320])
    </script>
--------------------OUTPUT-------------------------------
(4) ['dosa', 'palav', 'chapati', 'vada']
0
"dosa"
1
"palav"
2
"chapati"
3
"vada"
length
4
```

▼ Example 2

```
<script>
        var bike = new Array("r15","duke","rx 100","fz","mt");

        console.log(bike)
    </script>
```

▼ Using constructor.

▼ Syntax

```
var VarName = new Array(data1, data2, .......data N);
```

▼ Example

```

```

▼ Array Iteration in Forward direction (6 ways)

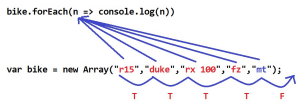
▼ 1st way (Normal)

```
<script>
        var bike = new Array("r15","duke","rx 100","fz","mt");
        console.log(bike)
    </script>
```

▼ 2nd way (for-loop)

```
<script>
        var bike = new Array("r15","duke","rx 100","fz","mt");
        for(i=0; i<bike.length; i++)
        {
            console.log(bike[i])
        }
</script>
----------------reverse----------
<script>
        var bike = new Array("r15","duke","rx 100","fz","mt");
        for(i=bike.length-1; i>=0; i++)
        {
            console.log(bike[i])
        }
</script>
```

▼ 3rd way (forEach loop)



▼ Example 1 (iterating every element)

```
<script>

      var bike = new Array("r15","duke","rx 100","fz","mt");
      bike.forEach(n => console.log(n))

</script>
```

```
--------------------------------
bike.forEach((n,i,num) => console.log(n,i,num))
```

▼ Example 2 (iterating every element, index value, object)

```
<script>

        var bike = new Array("r15","duke","rx 100","fz","mt");
        bike.forEach((n,i,num) => console.log(n,i,num))

  </script>
```

▼ Example 3 (Fetching element using index value)

```
<script>

        var bike = new Array("r15","duke","rx 100","fz","mt");
        bike.forEach((n,i) => {if(i==2){console.log(n)}})

  </script>
------------OUTPUT--------
rx 100
```

▼ Example 3 (Fetching index position using elements)

```
<script>

        var bike = new Array("r15","duke","rx 100","fz","mt");
        bike.forEach((n,i) => {if(n=="rx 100"){console.log(i)}})

  </script>
-----------OUTPUT--------
2
```

▼ 4th way (Using map)

▼ Example 1 (iterating every element)

```
<script>
        var bike = new Array("r15","duke","rx 100","fz","mt");
        bike.map(n => console.log(n))
</script>
```

▼ Example 4 (**iterating Object Array**)

```
<script>

        var manu = [{name:"manu",age:24,color:"red"},
                    {name:"Meghra",age:25,color:"white"},
                    {name:"Anand",age:1,color:"altra-white"}]
```

```
                manu.map(x => console.log(x.name))
</script>
----------OUTPUT-------------
```

▼ Example 2 (iterating element, index value, object)

```
<script>

        var bike = new Array("r15","duke","rx 100","fz","mt");
        bike.map((n,i,num) => console.log(n,i,num))

  </script>
```

▼ Example 3 (Fetching element using index value)

```
<script>

        var bike = new Array("r15","duke","rx 100","fz","mt");
        bike.map((n,i) => {if(i==4){console.log(n)}})

  </script>
------------OUTPUT--------
mt
```

▼ Example 3 (Fetching index position using elements)

```
<script>

        var bike = new Array("r15","duke","rx 100","fz","mt");
        bike.map((n,i) => {if(n=="mt"){console.log(i)}})

  </script>
------------OUTPUT--------
4
```

▼ 4th way (Using for-In loop)

```
<script>
        var bike = new Array("r15","duke","rx 100","fz","mt");
    for (const key in bike)
    {
        console.log(bike[key])
    }
</script>
```

▼ 5th way (Using Iterator)

```
<script>
var bike = new Array("r15","duke","rx 100","fz","mt");
```

```
for(const iterator of bike)
    {
        console.log(bike)
    }
</script>
```

▼ Array Inbuilt Functions (33 inbuilt functions)

▼ unshift()

- **unshift() function is used to add one or more element at the beginning of an existing Array.**

▼ **Example**

```
<script>
        var num = [1,2,3,4,5,6,8,8];
        document.write(num+"<br>")

        num.unshift(12,52,69)
        document.write(num)
    </script>
------------------OUTPUT--------------------------------
1,2,3,4,5,6,8,8
12,52,69,1,2,3,4,5,6,8,8
```

▼ shift()

- **shift() function is used to remove only one element at the beginning of an existing Array.**

▼ **Example**

```
<script>
        var num = [1,2,3,4,5,6,8,10];
        document.write(num+"<br>")
        document.write("shifted data==>"+num.shift()+"<br>")
        document.write(num)
    </script>
------------------OUTPUT-----------------------
1,2,3,4,5,6,8,10
shifted data==>1
2,3,4,5,6,8,10
```

▼ push()

- **push() is a function used to add one or more element at the end of an existing Array.**

▼ **Example**

```
<script>
        var num = [1,2,3,4,5,6,8,8];
```

```
        document.write(num+"<br>")

        num.push(12,52,69)
        document.write(num)
    </script>
-----------------------------------
1,2,3,4,5,6,8,8
1,2,3,4,5,6,8,8,12,52,69
```

▼ pop()

- **pop() function is used to remove only one element at the end of an existing Array.**

▼ Example

```
<script>
        var num = [1,2,3,4,5,6,8,10];
        document.write(num+"<br>")
        document.write("poped data==>"+num.pop()+"<br>")
        document.write(num)
    </script>
--------------------OUTPUT-----------------------------
1,2,3,4,5,6,8,10
poped data==>10
1,2,3,4,5,6,8
```

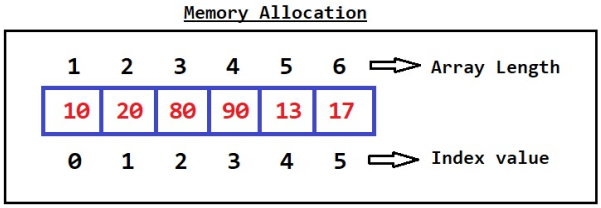
▼ slice()

- **slice() function is used to Extract the part of an Array elements from an existing Array and It will not change the original Array. It takes 2 arguments.**
  1. **Starting point**
  2. **Array length**

▼ Syntax

```
slice(Starting point, Array length);
```

▼ Example




```
<script>
        var x=[10,20,80,90,13,17];
```

```
        document.write("sliced date"+x.slice(3,5)+"<br>")
        document.write(x)
    </script>
------------------OUTPUT-------------------------
90,13
10,20,80,90,13,17
```

▼ splice()

- **splice() function is used to Add or Remove the elements from an existing Array and It will change the original Array. It takes 3 arguments**

```
splice (starting point , delete count, [add element]);
```

▼ Example (To remove the data)

```
<script>
        var x=[10,20,80,90,13,17];
        document.write(x+"<br>")

        document.write("spliced data "+x.splice(3,1)+"<br>")
        document.write(x+"<br>")
    </script>
-------------------OUTPUT----------------------
10,20,80,90,13,17
spliced data 90
10,20,80,13,17
```

▼ Example (To add the data)

```
<script>
        var x=[10,20,80,90,13,17];
        document.write(x+"<br>")

        x.splice(2,0,30,40,50) //here 0 means delete count
        document.write(x)
    </script>
----------------------OUTPUT----------------------
10,20,80,90,13,17
10,20,30,40,50,80,90,13,17
-----------------------
To add after 17 means last (6,0,20,30,40)
To add first (0,0,20,30,40)
```

▼ reverse()

- **reverse() function is used to reverse an existing Array.**

▼ Example

```
<script>
        var x=[10,20,80,90,13,17];
```

```
        document.write(x+"<br>")
        document.write(x.reverse())
    </script>
----------------OUTPUTY-------------------------------
10,20,80,90,13,17
17,13,90,80,20,10
```

▼ sort()

- **sort() function used to sort an existing Array in Ascending order.**

▼ Example (Ascending order)

```
<script>
        var x=[10,20,80,90,13,17];
        document.write(x+"<br>")
        document.write(x.sort())
    </script>
------------------------------
10,20,80,90,13,17
10,13,17,20,80,90
```

▼ Example (Descending order)

```
<script>
        var x=[10,20,80,90,13,17];
        document.write(x+"<br>")
        document.write(x.sort().reverse())
    </script>
----------------------OUTPUT--------------------------------
10,20,80,90,13,17
90,80,20,17,13,10
```

▼ Example (Remove Biggest element)

```
<script>
        var x=[10,20,80,90,13,17];
        document.write(x+"<br>")
        document.write(x.sort().pop()+"<br>")
        document.write(x)
    </script>
----------------OUTPUT-----------------------------------
10,20,80,90,13,17
90
10,13,17,20,80
```

▼ Example (Remove Smallest element)

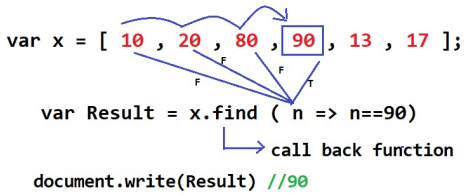
```
<script>
       var x=[10,20,80,90,13,17];
        document.write(x+"<br>")
        document.write(x.sort().shift()+"<br>")
        document.write(x)
```

```
    </script>
------------------OUTPUT-------------------------
10,20,80,90,13,17
10
13,17,20,80,90
```

▼ find()

- **find() is a call back function which is used to find the elements from an existing array based on passed condition.**

▼ Example




```
<script>
        var x=[10,20,80,90,13,17];

        var Result = x.find (n => n==90)

        document.write(Result+"<br>")
        if(Result==90)
        {
            document.write("90 is present in an array")
        }else{
            document.write("90 is not present in an array")
        }
    </script>
------------------OUTPUT---------------------------------
90
90 is present in an array
```

▼ filter()

- **filter() is call back function which is used to filter the elements based on passed condition.**

▼ Example

```
<script>
        var x=[10,20,80,90,13,17];

        var Result = x.filter (n => n>75)

        document.write(Result)
    </script>
```

```
------------------OUTPUT----------------------
80,90
```

▼ **concat()**

- **concat() function is used to merge two Arrays.**

▼ **Example**

```
<script>

        var x=[10,20,80,90,13,17];
        document.write(x+"<br>")
        var x1 = [100,200,300];
        document.write(x1+"<br>")

        document.write(x.concat(x1))

</script>
------------------OUTPUT--------------------------------
10,20,80,90,13,17
100,200,300
10,20,80,90,13,17,100,200,300
```

21. **String methods and Properties in JS ?**

▼ **Ans**

▼ **slice()**

- **slice() function is used to Extract the part of the string from an existing String and returns the extracted String in new String. It takes 2 arguments**
  1. **start position**
  2. **end position (end not included)**
- **If arguments are Negative, then position is counted from end of the String.**

▼ **Syntax**

```
slice(start,end)
```

▼ **Example**

```
<script>
        let str="Apple, Banana, Kiwi";
        let part = str.slice(7,13)
        document.write(part)
    </script>
-------------OUTPUT--------------------------
Banana
```

▼ **Example 2**

```
<script>
        let str="Apple, Banana, Kiwi";
        let part = str.slice(-12,-6)
        document.write(part)
    </script>
-----------------OUTPUT--------------------------------
Banana
```

▼ **substring()**

- **substring() function is used to Extract the part of the string from an existing String and returns the extracted String in new String. It takes 2 arguments**
  1. **start position**
  2. **end position (end not included)**
- **And It do not takes negative value.**

▼ **Syntax**

```
substring(start,end)
```

▼ **Example**

```
<script>
        let str="Apple, Banana, Kiwi";
        let part = str.substring(7,13)
        document.write(part)
    </script>
--------------OUTPUT--------------------------
Banana
```

▼ What is difference between slice() and substring ?

▼ **substr()**

- **substr() function is used to Extract the part of the string from an existing String and returns the extracted String in new String. It takes 2 arguments**
  1. **Start position**
  2. **Length of the String which has to be extracted.**

▼ **Syntax**

```
substr(star,length)
```

▼ **Example**

```
<script>
        let str="Apple, Banana, Kiwi";
        let part = str.substr(7,6)
        document.write(part)
    </script>
------------------OUTPUT-----------------------------
Banana
```

▼ includes()

- **includes() function is used to check whether the particular specified String value present in an existing String or not. It return boolean true or false value.**
- **It takes 2 argument,**
  1. **Search value**
  2. **Start Position (by default it is 0)**

▼ **Syntax**

```
string.includes(searchvalue,start)
```

▼ **Example**

```
<script>
        let str="Hello world, welcome to the universe.";
        document.write(str.includes("world")+"<br>")
        document.write(str.includes("world",8));
    </script>
---------------------OUTPUT-----------------------
true
false
```

▼ replace()

- **replace() function is used to replace an existing String value to new String value.**

▼ **Example**

```
<script>
        let text="Please visit Microsoft";
        document.write(text+"<br>")
        let newtext = text.replace("Microsoft","jSpider")
        document.write(newtext)
    </script>
--------------------OUTPUT-------------------------------
Please visit Microsoft
Please visit jSpider
```

▼ repeat()

- **repeat() function is used to repeat the particular specified String value based on number which we pass.**

▼ Example

```
<script>
        var str="java script";
        var x=str.repeat(6);
        console.log(x)
    </script>
------------------OUTPUT--------------------------------------
java script
java script
java script
java script
java script
java script
```

▼ trim()

- **trim() is used to remove the white spaces from the both Ends of the String value.**

▼ Example

```
<script>
        let text1="     Hello World!     ";
        let text2= text1.trim();
        document.write("Length text1= "+text1.length+"<br>")
        document.write("Length2 text2= "+text2.length)
    </script>
-----------------OUTPUT----------------------
Length text1= 22
Length2 text2= 12
```

▼ length()

- **length() function is used to find the length of the String.**

▼ Example

```
<script>
    let txt="abcdefghij"
    let length=txt.length;
    console.log(length)
    </script>
----------------OUTPUT--------------------
10
```

▼ indexOf()

- **indexOf() function is used to fetch the index value of first occurrence of the particular specified String value. If specified String value is not present it returns -1.**

  ▼ Example

```
<script>
        let str="Please locate where 'locate' occurs!";
        str.indexOf("locate");
        str.indexOf("kocate");
        document.write(str.indexOf("locate")+"<br>")
        document.write(str.indexOf("kocate"))
    </script>
--------------OUTPUT--------------------------
7
-1
```

▼ lastIndexOf()

- **lastIndexOf() function is used to fetch the index value of last occurrence of particular specified String value.**

  ▼ Example

```
<script>
        let str="Please locate where 'locate' occurs!";
        str.indexOf("locate");
        str.indexOf("kocate");
        document.write(str.lastIndexOf("locate")+"<br>")
        document.write(str.lastIndexOf("kocate"))
    </script>
-------------------OUTPUT-------------------------------------
21
-1
```

▼ charAt()

- **charAt() function is used to fetch the character from the particular specified index position in the String.**

  ▼ Example

```
<script>
        var str="hi guys how are you all";
        var x=str.charAt(3);
        console.log(x)
        document.write(x)
    </script>
----------------OUTPUT-------------------------
g
```

▼ charCodeAt()

- **charCodeAt() function is used to fetch ASSCI value of the character from the particular specified index position in the String.**

▼ **Example**

```
<script>
        var str="java script";
        var x=str.charCodeAt(1);
        console.log(x)
        document.write(x)
    </script>
-----------------OUTPUT------------------------------
97
```

22. **What is Object ?**

▼ **Ans**

- **Object is Real World Physical Entities which has its own States and Behaviors. In JS Object Contains Key and Value pair.**
  1. **State Represents the Properties of Object**
  2. **Behavior Represents the Functionality of an Object**
- **In JS, We can create Object in 3 ways : -**

▼ **Object Literals**

- **Object literals is used to add and fetch the data using key. values can be duplicate but key cannot be duplicate.**

▼ **Syntax**

```
var ObjectName = { key1 :"value1",
                   key2 :"value2",
                      .
                      .
                      .
                    keyN :"valueN"
                  }
```

▼ **Example 1**

```
<script>
        var person={
            name:"manu",
            age:29,
            empid:"typ8024",
            hobbies:["sports","teaching","dancing"],
            add:{
                state:"karnataka",
                city:"banglore",
                area:"btm layout"
```

```
            }
        }
        console.log(person)
        console.log(person.empid)
        console.log(person.hobbies[1])
        console.log(person.add.state)
        person.email="manuk969@"
        person.add.pincode=583131
    </script>
```

▼ Example 2 (Storing of multiple object without using class)

```
<script>
        var emp=[
            {
                name:"sindhu",
                sal:1200325,
                company:"IBM"
            },
            {
                name:"indhu",
                sal:12325,
                company:"TCS"
            }
            ,
            {
                name:"bindhu",
                sal:1254325,
                company:"wipro"
            }
        ]
        console.log(emp)
        console.log(emp[2].sal)
    </script>
```

▼ Using Class

▼ Syntax

```
var VariablenName = new className();
```

▼ Example 1

```
<script>
        class car
        {
            color="red"
            price=120000
            name="tesla"
            details=function()
            {
                document.write("good car")
            }
        }
        var c1=new car();
```

```
        console.log(c1)
    </script>
```

▼ Example 2

```
<script>
        class car
        {
            color="red"
            price=120000
            name="tesla"
            details=function()
            {
                document.write("good car")
            }
        }
        var c1=new car();
        console.log(c1)
        console.log(c1.color)
        console.log(c1.details())
    </script>
```

▼ Using Constructor

- **Constructor is a member of class which is used to initialize the instance variables.**

▼ Syntax

```
constructor()
{

}
```

▼ Example

```
<script>
        class car
        {
            constructor()
            {
                document.write("i am construcotr"+"<br>")
            }
        }
        var c1=new car(); //constructor invocation
        var c2=new car(); //constructor invocation
        var c3=new car(); //constructor invocation
    </script>
------------------OUTPUT------------------------------
i am construcotr
i am construcotr
i am construcotr
```

- **If we want to Initializing the instance variable inside constructor to get dynamic output. We must use 'this' keyword**

  ▼ **'this' keyword**

  - **Current invoking object reference address.**

  ▼ **Example**

```
<script>
        class car
        {

            constructor(c,p,b)
            {
              this.color=c;
              this.price=p;
              this.brand=b;
              document.write(this.color+" "+this.price+" "+this.brand+"<br>")
            }
        }
        var c1=new car("red",12000,"tesla"); //constructor invocation
        var c2=new car("blue",13000,"musta"); //constructor invocation
        var c3=new car("black",16551,"punto"); //constructor invocation
    </script>
------------------OUTPUT----------------------------
red 12000 tesla
blue 13000 musta
black 16551 punto
```

  ▼ **Example**

```
<script>
        class car
        {

            constructor(c,p,b)
            {
              this.color=c;
              this.price=p;
              this.brand=b;
              document.write(this.color+" "+this.price+" "+this.brand+"<br>")
            }
        }
        var c1=new car("red",12000,"tesla"); //constructor invocation
        var c2=new car("blue",13000,"musta"); //constructor invocation
        var c3=new car("black",16551,"punto"); //constructor invocation
    </script>
--------------------OUTPUT----------------------------
red 12000 tesla
blue 13000 musta
black 16551 punto
```

19. **What is the super most object in JavaScript ?**

  ▼ **Ans**

- **Window**
- **f —> function**
- **: or = —> properties**
- **{ } —> Object**

20. **What is JSON ?**

▼ Ans

- **JSON stands for Java Script Object Notation. Json format is used to add data to the database and fetch the data from the database. Json is the only way we can add data to any database.**

▼ Syntax

```
var VarName = { "key1" :"value1",
                "key2" :"value2",
                      .
                      .
                      .
                "keyN" :"valueN"
                  }
```

- **In JS, We can convert object literals into a Json format in 2 ways :**
  1. **with Internet (Json formatter and validation)**
  2. **without Internet (Using Stringifly() function)**

▼ What is Stringifly() function ?

- **Stringifly() is a function present inside Json object which is used to convert the object literals data into Json format.**

▼ Example

```
<body>
    <script>
        var person= {
            name:"Manu",
            habbies:["Sports","Teaching","Dancing"],
            add:{
                state:"Karntaka"
            }
        }
        console.log(person)
        var result=JSON.stringify(person)
        console.log(result)
    </script>
</body>
--------------OUTPUT-----------------------------
{"name":"Manu","habbies":["Sports","Teaching","Dancing"],"add":{"state":"Karntaka"}}
```

25. **What is setTimeout Function ?**

▼ **Ans**

- **SetTimeout Function allows us to run a function once after given interval of time. It takes parameter as Function, Delay Time in milisec and arguments. It returns timer ID.**
- **We use setTimeout Function, When we want to execute our JS code after some time.**
- **Syntax →** `let timerID = setTimeout (function,<delay>,<arg1>,<arg2> , , ,)`

▼ **ClearTimeout is used to cancel the execution. (In case, If we change our mind)**

```
<script>
  let timeID = setTimeout(()=>{alet("never"),1000})
  clearTimeout(timeID) //cencel the Execution
</script>
```

▼ **Example 1**

```
<script>
    console.log("Start")
    setTimeout(()=>{
                console.log("Hi ")
                    },2000)

    console.log("Asynchronous Execution 1");
    </script>
--------------OUTPUT------------
Start
Asynchronous Execution 1
Hi
```

▼ **Example 2**

```
<script>
    console.log("Start")
     let a = setTimeout(()=>{
                console.log("Hi")
                    },2000)
         clearTimeout(a)
    console.log("Asynchronous Execution 1");
    </script>
--------------OUTPUT------------
Start
Asynchronous Execution 1
```

▼ **Example 3**

```
<script>
     console.log("Start")
     let a = (x)=>{console.log("HI " + x);}
     setTimeout(a , 2000 , 2)
```

```
     console.log("End");
</script>
--------------OUTPUT------------
Start
End
HI 2
```

26. **What is setInterval Function ?**

▼ **Ans**

- **SetInterval Function allows us to run a function not only once, but regularly after the given interval of time. To stop further calls, we can use clearInterval (timerID) function.**
- **We use setInterval Function, When we want to execute our JS code again an again after a set period of time.**
- **Syntax →** `let timerID = setInterval (function,<delay>,<arg1>,<arg2> , , ,)`

▼ **Example 1**

```
<script>
    console.log("Start")
    setInterval(()=>{
                console.log("Hi ")
                    },1000)

    console.log("End");
    </script>
--------------OUTPUT------------
Start
End
Hi
Hi //This will print every 1 second
```

▼ **Example 2**

```
<script>
    console.log("Start")
   let a = setInterval(()=>{
                console.log("Hi ")
                    },1000)
                    clearInterval(a)
    console.log("End");
    </script>
--------------OUTPUT------------
Start
End
```

25. **What is Asynchronous and Synchronous Programming ?**

▼ **Ans**

- **A Synchronous programming allows single execution happen one at a time.**

  ▼ Example

```
<script>
    console.log("Synchronous Execution")
    for (let i = 0; i < 5; i++)
    {
        console.log(i)
    }
    console.log("Synchronous Execution End");
    </script>
-----------------OUTPUT--------------
Synchronous Execution
1
2
3
4
Synchronous Execution End
```

- **An Asynchronous programming allows multiple execution happen at the same time.**

  ▼ Example

```
<script>
    console.log("Asynchronous Execution")
    setTimeout(()=>{
                    for (let i = 0; i < 5; i++)
                        {
                            console.log(i)
                        }
                    },2000)
    console.log("Asynchronous Execution 1");
    </script>
-----------------OUTPUT--------------
Asynchronous Execution
Asynchronous Execution 1
1
2
3
4
```

25. **What is Promise ?**

▼ Ans

- **The solution to the call back is promise. A Promise is a "Promise of code execution" The code either gets executes or fails, in both the cases the subscriber will be notified by using .then() and .catch() method.**

▼ Syntax

```
let promise = new Promise(function(resolve,reject)
                                    {
                                        //Execute
```

```
                                    }
                            )
```

- **resolve and reject are two callbacks provided by JS itself.**
    1. **resolve(value) -> If the job is finished successfully.**
    2. **reject(error) -> If the job fails.**
- **The promise object returned by the new Promise constructor has 2 properties**
    1. **states -> Initially undefined, then changes to either "fulfilled" when resolve is called or "rejected" when reject is called.**
    2. **result -> Initially undefined, then changes to values if resolved (resolve(value)) or error when rejected (reject(error)).**
- **.then() → When a promise is successful, We can use the resolved data.**
- **.catch() → When a promise fails, We can catch the error, and do something with the error information**

▼ Example 1 (Promise)

```
<script>
        let p = new Promise((resolve, reject)=>{
            let a = 1 + 2
            if(a == 2)
            {
                resolve('Sucess')
            }
            else
            {
                reject('Failed')
            }
        })
        p.then((x)=>{console.log(x);})
         .catch((x)=>{console.log(x);})
    </script>
------------------OUTPUT------------------
Failed
```

▼ Example 2 (Difference between callbacks and Promise)

▼ callbacks

```
<script>
        const a = false
        const b = false

        function manu(x, y)
        {
            if(a)
            {
                y({name:'Charvi',age:'22'})
            }
```

```
            else if(b)
            {
                y({name:'priya',age:'22'})
            }
            else
            {
                x('Monk Mode')
            }
        }
        manu((w)=>{console.log(w);},
             (e)=>{console.log(e.name+" "+e.age);})
    </script>
</body>
------------OUTPUT---------------
Monk Mode
```

▼ **Promise**

```
<script>
        const a = false
        const b = false

        function manu()
        {
        return new Promise((resolve, reject)=>{
            if(a)
            {
                reject({name:'Charvi',age:'22'})
            }
            else if(b)
            {
                reject({name:'priya',age:'22'})
            }
            else
            {
                resolve('Monk Mode')
            }
            })
        }
        manu().then((w)=>{console.log(w);})
              .catch((e)=>{console.log(e.name+" "+e.age);})
    </script>
------------OUTPUT---------------
Monk Mode
```

26. **What is Closure ?**

▼ **Ans**

- **The Closure is a binding of parent function variable with the child function. This enables the JS program to use parent function number inside child function.**

▼ **Example**

```
function Person(){
        let age = 21;
        function A(){
```

```
            return age
        }
        return ;
    }

let b = Person();
console.log(b());
```

▼ **Note**

- **The biding of child/inner/nested function with its lexical environmental(parent/ function state) is known as Closure.**

▼ **Points to remember on Closure**

- **Closure helps to achieve scope chain(lexical environment) from child function to parent function.**
- **Closure preserves the state of parent function even after the execution of thier parent function is compiled.**
- **A function will have reference to closure. for every a parent function , new closure will going to created.**
-

- **Disadvantages of closure —> High memory consumption.**