**JAVA SCRIPT**

**Content Page No.**

1. Operators 02
2. JavaScript Variables 04
3. JavaScript Data types 05
4. Functions 14
5. Network calls (Synchronous & Asynchronous calls) 21
6. CallBacks 22
7. Promises 25
8. Closures 32

2022/11/17

The document.write() method **writes a string of text to a document stream opened by document.open()** . Note: Because document.write() writes to the document stream, calling document.

**Example1:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Java Script Basics</title>

</head>

<body>

<script>

document.write("Welcome to JavaScript"); // Welcome to JavaScript

document.write(10+10); //20

document.write(10+"10"); //1010

document.write(10+ +"10"); //20

document.write(1+ +"2"+ +"2"); //5

document.write(10+5/10-5); //5.5

document.write((10+5)/(10-5)); //3

-, \*, / string 🡪 number

document.write(10-"10"); //0

document.write(10\*"10"); //100

document.write(10/"10"); //1

document.write(10/"0"); //Infinity

document.write(0/0); //NaN

document.write(0/"0"); // NaN Stands for Not a Number

document.write(0/"A"); //NaN

**true --- 1, false ---0**

document.write(true+true); //2

document.write(1+true); //2

document.write(1+true+1+false); //3

document.write(1+ "true"); //1true

document.write(true/false); //Infinity

**Ternary Operator**

**Syntax:**

**condition? true statement: false statement**

true? document.write("Hello") : document.write("Bye"); //Hello

false? document.write("Hello") : document.write("Bye"); //Bye

9>8? document.write("Hello") : document.write("Bye"); //Hello

9<8? document.write("Hello") : document.write("Bye"); //Bye

document.write(9>8>7); //false

Executin starts from left to right 9>8= ture>7 =1>7 =false

document.write(1<2<3); // 1<2= true<3 =1<3 =true

document.write(3<2<1); // 3<2= false<1 =0<1 =true

**= (Assignment operator)**

**== (Comparision operator)**

**=== (Strict comparision)**

== operator will compare left operand value with right operand value

=== operator will compare left operand value with datatype to right operand value with datatype

document.write(10=="10") ; //true

document.write(10==="10") ; //false

document.write("10"==="10"); //true

document.write(0.1+0.2); //0.30000000000000004

**Java Script is more accurate while adding fractional numbers**

document.write(0.1+0.2 ==0.3); //false

document.write(0.1+0.2 === 0.3); //false

**& (And) if both are true, then result is true**

**| (Or) if at least one is true, then result is true**

**^ (XoR) if both are opposite, then result is true**

document.write(1&1); //1

document.write(1&0); //0

document.write(0&1); //0

document.write(0&0); //0

document.write(1|1); //1

document.write(1|0); //1

document.write(0|1); //1

document.write(0|0); //0

document.write(1^1); //0

document.write(1^0); //1

document.write(0^1); //1

document.write(0^0); //0

console.log("Debugging soon......!");

console.table([10,20,30,40]);

</script>

</body>

</html>

2022/11/18

**Variables**

Variable are used to store data

**Ex:**

string

number

boolean

arrays

objects

--

--

we will declare variables with the help of

1) var

2) let

3) const

4) nothing (no declaration)

let and const are introduced in ES6 version (ECMA Script)

Variables declaration should contains a-z, A-Z, 0-9, $ and \_

Variables declaration should not starts with

**Syntax:**

var/let/const/nothing variable\_name = value

**Ex:**

var msg = "welcome to javascript";

**javascript supports 7 datatypes**

1) string

2) number

3) boolean

4) undefined

5) null

6) bigint

7) symbol

**string**

collection of "characters" called as "string"

we can declare variable in 3 ways

1) "" (double quotes)

2) '' (single quotes)

3) `` (back tick)

``(back tick) called as "temporary literal"

``(back tick) operator introduced in "ES6"

``(back tick) opearator used to define the "paragraphs"

**number**

javascript supports following types of numbers

1) decimal

2) double

3) hexadecimal (should starts with 0x) (hexadecimal number may contain A-F)

4) octal (should starts with 0o)

5) binary (should starts with 0b) (binary number should contain only '0' and '1')

**boolean**

javascrpit supports following boolean values

1) true --------------- 1

2) false -------------- 0

**undefined**

undefind is the datatype in javascript

variable decalred but value assigned called as undefined

**null**

null also datatype in javasript

making "un-used members" availability to garbage collector called as "null" (clearing the browser memory is called null)

**bigint**

bigint used to store the large numbers

bigint introduced in "ES6" version

bigint numbers suffix with "n";

range of bigint is 2^53-1 to -2^53-1

**Symbol**

it is used to represent the data uniquley

symbol also introduced in ES6 vesion.

**Example 1:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Variables</title>

</head>

<body>

<script>

var sub\_one=`React JS`;

var sub\_two=`Node JS`;

var sub\_three=`MangoDB`;

var mern\_stack1=`React JS <==> Node JS <==> MangoDB`;

var mern\_stack=`${sub\_one} ,,,,, ${sub\_two} <==> ${sub\_three},,, ${mern\_stack1}`;

document.writeln(mern\_stack1);

document.writeln("<br><br>");

document.writeln(mern\_stack);

document.writeln("<br><br>");

var decimal\_number= 100;

var double\_number=100.02335;

var hexadecimal\_number=0x123ABC;

var octal\_number= 0o123;

var binary\_number=0b1011;

document.writeln(decimal\_number,"<br>",

double\_number, "<br>",

hexadecimal\_number,"<br>",

octal\_number,"<br>",

binary\_number);

document.write("<br><br>");

Output:

100

100.02335

1194684

83

11

javascript internally converts hexadecimal, octal, binary to decimal numbers

var flag=true;

var flag1=false;

flag ? document.write("true block") : document.write("false black");

document.write("<br>");

flag1 ? document.write("true block") : document.write("false black");

document.write("<br><br>");

var x;

document.writeln(x); // undefined

document.writeln("<br><br>")

x=null;

document.writeln(x); //null

document.writeln("<br><br>")

var large = 1111112789812222391731097777777777777777777777777791723017230713071207317n;

document.writeln(large);

document.write("<br><br>");

var r101=Symbol(100);

var r102=Symbol(100);

console.log(r101); //Symbol(100)

console.log(r102); //Symbol(100)

document.write(r101, r102);

document.write("<br><br>");

**type of operator**

document.write(typeof "Hello"); //string

document.write("<br>");

document.writeln(typeof 100); //number

document.write("<br>");

document.write(typeof true) //boolean

document.write("<br>");

document.write(typeof undefined) //undefined

document.write("<br>");

document.write(typeof null) //object

document.write("<br>");

document.write(typeof 100n); //bigint

document.write("<br>");

document.write(typeof Symbol(100)); //symbol

document.write("<br>");

document.write(typeof []); //object

document.write("<br>");

document.write(typeof {}); //object

document.write("<br><br>");

var arr1=[10, 20, 30, 40, 50];

to iterate arrays we have advanced loops

forEach()

for...of()

function without name called as anyonymus function

arr1.forEach(function(element, index){

document.write(element," ---- ",index);

document.write("<br><br>");

});

for(var v1 of arr1)

{

document.write(v1);

document.write("<br>");

}

**JSON**

JSON stands for JavaScript Object Notation

JSON used to transfer the data over the network

JSON is light weight

Objects ---- {}

Arrays ---- []

data ---- key and value pairs

key and value saparated by using " : "

key and value pairs saparated using " , "

var obj={

key1 : "ReactJS",

key2 : "NodeJS",

key3 : "MangoDB"

};

document.write(obj.key1, obj.key2, obj.key3 );

</script>

</body>

</html>

2022/11/19

**Example 2:**

<!DOCTYPE html>

<html>

<head>

<title>Variables</title>

</head>

<body>

<script>

var x=100;

var x=200;

document.write(x);

var : 200

let x=100;

let x=200;

document.write(x);

//let : Uncaught SyntaxError: Identifier 'x' has already been declared

//var keyword allows the duplicate variables

//let keyword wont allows the duplicate variables

</script>

<script>

document.write(x);

let x=100;

//var : undefined

//let : Uncaught ReferenceError: Cannot access 'x' before initialization

//accessing variable before declaration and initilization with var keyword called as variable hoisting

//variable hoisting is raised with var keyword

//we can overcome variable hoisting with let keyword

</script>

<script>

function func\_one()

{

{

var x=100;

let y=200;

}

document.write(x); //100

document.write(y); //variables1.html:40 Uncaught ReferenceError: y is not defined at func\_one

}

func\_one();

// var members are functional scope members

// let members are block scope members

</script>

<script>

let x=100;

{

let x=200;

}

document.write(x);

//var : 200

//let : 100

//global polluting issue raised because of var keyword

//we can overcome global polluting issue with let keyword

</script>

**var let**

1. var keyword introduced in ES1 version 1) let keyword introduced in ES6
2. duplicate variables are allowed 2) duplicate variable are not allowed
3. variable hoisting is raised because of 3) we can overcome variable hoisting

var keyword with let keyword

1. global polluting issue is 4) global polluting issue we can overcome

raised because of var keyword with let keyword

1. var members are functional scope members 5) let members are block scope members

**const**

const is the keyword introduced in ES6 version

const keyword used to declare the variables

re-initalization not possible with const keyword

<!-- <script>

const x=100;

x=200;

//Uncaught TypeError: Assignment to constant variable.

const arr1=[10, 20, 30, 40, 50];

arr1 = [];

//Uncaught TypeError: Assignment to constant variable.

// arr1 = [100, 200, 300, 400, 500];

//Uncaught TypeError: Assignment to constant variable.

arr1[0] = 100;

arr1[4] = 500;

arr1[5] = 600;

arr1[7] = 800;

document.write(arr1); //100,20,30,40,500,600

const obj = {

key1 : "ReactJS",

key2 : "NodeJS",

key3 : "MangoDB"

};

//obj = {};

//Uncaught TypeError: Assignment to constant variable.

obj = {key1 : "ReactJS with TypeScript",

key2 : "NodeJS with TypeScript",

key3 : "MangoDB with cloud" };

//Uncaught TypeError: Assignment to constant variable.

obj.key1 = "ReactJS with TypeScript";

obj.key2 = "NodeJS with TypeScript";

obj.key3 = "MangoDB with cloud";

document.write(JSON.stringify(obj));

{"key1":"ReactJS with TypeScript","key2":"NodeJS with TypeScript","key3":"MangoDB with cloud"}

</script>

<script>

x=100;

document.write(x); //100

</script>

</body>

</html>

2022/11/20

**Functions**

Particular business logic called as function

Functions are used to reuse the business logics

We have 3 types of functions

1. named functions
2. anonymous functions
3. arrow functions

**named functions**

The function with user defined name called as named functions

**Syntax:**

**function defination**

function function\_name(parameter1, parameter2, parameter3,...parametern){

business logic;

}

**functions calling**

function\_name(argument1,argument2,argument3,.....argumentn)

**Ex:**

<!DOCTYPE html>

<html>

<head>

<title>Named Functions</title>

</head>

<body>

<script>

function func\_one(){

document.write("Welcome to named functions <br><br>");

}

func\_one();

func\_one();

func\_one();

</script>

<script>

function func\_one(){

document.write("welcom to named functions <br><br>");

}

document.write(func\_one) ; //function defination

</script>

<script>

function func\_one(parameter1, parameter2, parameter3){

document.write(parameter1, parameter2, parameter3);

}

func\_one("Hello Darling ", "Hai Gorgeous ", "Looking Beautiful");

//Hello Darling Hai Gorgeous Looking Beautiful

func\_one(100 , 200 , 300 , 400); //100 200 300

func\_one(); //undefinedundefinedundefined

func\_one(undefined, "Hai Beautiful"); //undefined Hai Beautiful undefined

func\_one(null,null,null,100); //null null null

</script>

<script>

function func\_one(){

return 100;

}

let x=func\_one();

document.write(x \* x); //10000

</script>

<script>

function func\_one(){

return 1&1? "Hello" : "Bye";

}

let result=func\_one();

document.write(result); //Hello

</script>

<script>

function func\_one(parameter1){

return parameter1 \*parameter1;

}

let res1=func\_one(10);

document.write(res1 \* res1); //10000

</script>

<script>

// ... called as spread operator

// spread operator introduced in ES6 vesion

// defalut value of spread operator is [] (Empty array)

function func\_one(...parameter1){

document.write(parameter1);

}

func\_one(10); //[10]

func\_one(10, 20, 30); //[10,20,30]

func\_one("Katherine", " Margott", " Gal Gadot "); //[Katherine, Margott, Gal Gadot]

func\_one(); //[]

func\_one(undefined, undefined); //[undefined, undefined] [,]

func\_one(null, null, undefined); //[null, null, undefined] [, ,]

</script>

<script>

function func\_one(...parameter1, ...parameter2){

}

**Note:** we can pass only one spread operator per function

</script>

<script>

function func\_one(...parameter2, parameter1){

}

//Note: spread operator occurance always last in parameters

</script>

<script>

function func\_one(parameter1, ...parameter2){

document.write(parameter1, parameter2);

}

func\_one(); //undefined

func\_one(100,100); //100 100

func\_one(100, 200, 300); //100 [200,300]

func\_one(undefined, undefined); //undefined [undefined]

func\_one(nul, 100, null); //null [100, null]

//Uncaught ReferenceError: nul is not defined

</script>

</body>

</html>

2022/11/21

<script>

//while defining the function we will intilize the parameters

// this concept is called default parameters in function

// Introduced in ES6

function func\_one(parameter1="Hello"){

document.write(parameter1);

}

func\_one(); // Hello

func\_one("Welcome"); // Welcome

func\_one(undefined); // Hello

func\_one(null); // null

</script>

<script>

function func\_one(parameter1, parameter2="Hello2", parameter3="Hello3", ...parameter4){

document.write(parameter1, parameter2, parameter3, parameter4);

}

func\_one(); //undefined Hello2 Hello3 []

func\_one("Hello1", undefined, undefined, "Hello4"); //Hello1 Hello2 Hello3 [Hello4]

func\_one(undefined, undefined, undefined, undefined); //undefined Hello2 Hello3 [undefined]

func\_one(null, null, null, null); //null null null [null]

</script>

<script>

Anonymous function

The function without name called as Anonymous function

var x=function(){

document.write("Welcome to anonymous function");

}

x(); // Welcome to anonymous function

</script>

<script>

let x=function(parameter1){

document.write(parameter1);

}

x("Hello"); //Hello

x(); //undefined

x(null); //null

</script>

<script>

// Rule1: Create the function

// Rule2: call the function

func\_one(); //Hello

//Uncaught TypeError: func\_one is not a function

function func\_one(){

document.write("Hello");

}

var func\_one = function(){

document.write("Hello");

}

// function hoisting is possible in named function

// function hoisting is not possible in anonymous function

</script>

Arrow function

Arrow functions introduced in ES6 version

we will represent Arrow Function with =>

<script>

var x = () =>{

document.write("Welcome to arrow function");

}

x(); //Welcome to arrow function

</script>

<script>

let x=(parameter1, parameter2, parameter3)=>{

document.write(parameter1, parameter2, parameter3);

}

x("Katherine ", " Margott", " Gal Gadot"); //Katherine Margott Gal Gadot

</script>

<script>

let x = () =>{

return "welcome to arrow function";

}

let res=x();

document.write(res); //welcome to arrow function

</script>

<script>

let x=() => "welcome to arrow function";

let res=x();

document.write(res); //welcome to arrow function

</script>

<script>

let x=(parameter1) => parameter1 \* parameter1;

let res=x(10);

document.write(res); //100

//short hand functions

</script>

<script>

setTimeout(function(){

document.write("Hello");

} ,5000);

//Hello

//setTimeout is used to execute Anonymous funtion or Arrow function after some time

</script>

<script>

setTimeout(()=>{

document.write("Hello");

},5000);

</script>

<script>

let x=setInterval(function() {

document.write(new Date().toLocaleString()+"<br>");

}, 1000);

//setInterval is used to execute function continously

clearTimeout(x);

//clearTimeout is used to break the setInterval execution

</script>

2023/04/19

We have two types of network class

1) Synchronous

2) Asynchronous

* Execution of network calls one by one called as synchronous calls.
* Execution of network calls parallelly called as asynchronous calls

We can make asynchronous call in different ways

1) callbacks

2) promises

3) observables

---etc.

**CallBacks**

Passing one function definition to another function parameter called as callback.

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>callBacks</title>

</head>

<body>

<script>

function fun\_one(param1){

param1();

}

fun\_one(

function fun\_two(){

document.write("welcome to callback");

}

);

//welcome to callback

</script>

<script>

let func\_one = (param1, param2, param3) =>{

param1();

param2();

param3();

};

func\_one(()=>{

document.write("ReactJS");

}, ()=>{

document.write("NodeJS");

}, ()=>{

document.write("MongoDB");

});

//ReactJS NodeJS MongoDB

</script>

<script>

let fun\_one=(param1)=>{

return param1("Hello");

}

fun\_one((x)=>{

document.write(x);

});

//Hello

</script>

<script>

let func\_one=(param1)=>{

return param1(100, 200, 300);

}

func\_one((res1, res2, res3)=>{

document.write(res1, res2, res3);

});

//100 200 300

</script>

</body>

</html>

**CallBack Hell**

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Call Back Hell</title>

</head>

<body>

<script>

let add=(num, callback)=>{

callback(num+5, false);

}

let sub=(num, callback)=>{

callback(num-3, false);

}

let mul=(num, callback)=>{

callback(num\*2, false);

}

let div=(num, callback)=>{

callback(num/2-2, false);

}

add(5,(addRess, error)=>{

if(!error)

{

// console.log(addRess);

sub(addRess, (subRes, error)=>{

if(!error)

{

// console.log(subRes);

mul(subRes, (mulRes, error)=>{

if(!error)

{

// console.log(mulRes);

div(mulRes, (divRes, error)=>{

if(!error)

{

console.log(divRes);

}

})

}

})

}

})

}

});

</script>

</body>

</html>

* dependencies between callbacks called as callback hell
* callbacks never recomended in real time.
* Alternative solutions or callback hell is Promises

**PROMISES**

* Promises are used to make Asynchronous calls
* promises are special JavaScript Objects
* Promises have 3 States

1) Pending

2) Resolve

3) Reject

**CREATING THE PROMISE**

Promise() object is the predefined class, used to create the Promises

**CONSUME THE PROMISE**

We will consume Promise in two ways

1) then()

2) async & await

<!DOCTYPE html>

<html lang="en">

<head>

<title>Promises</title>

</head>

<body>

<script>

let promise\_1=new Promise((resolve, reject)=>{

resolve("welcome to promise in javascript ");

});

// promise\_1.then((posRes)=>{

// document.write(posRes);

// },(errRes)=>{

// document.write(errRes);

// });

async function consume(){

let res= await promise\_1;

document.write(res)

}

consume();

</script>

<script>

let promise1=new Promise((resolve, reject)=>{

setTimeout(()=>{

resolve("Hello\_1");

},0);

});

// let promise2=new Promise((resolve, reject)=>{

// setTimeout(()=>{

// resolve("Hello\_2");

// },5000);

// });

let promise2=new Promise((resolve, reject)=>{

setTimeout(()=>{

reject("Call Failed");

},5000);

});

let promise3=new Promise((resolve, reject)=>{

setTimeout(()=>{

resolve("Hello\_3");

},10000);

});

// promise1.then((posRes)=>{

// document.write(posRes);

// },(errRes)=>{

// document.write(errRes);

// });

// promise2.then((posRes)=>{

// document.write(posRes);

// },(errRes)=>{

// document.write(errRes);

// });

// promise3.then((posRes)=>{

// document.write(posRes);

// },(errRes)=>{

// document.write(errRes);

// });

// Promise.all([promise1, promise2, promise3])

// .then((posRes)=>{

// document.write(posRes);

// }, (errRes)=>{

// document.write(errRes);

// });

//Hello\_1,Hello\_2,Hello\_3

// all() function is used to overcome the data redundancy (used to consume all functions at a time)

// Promise.all([promise1, promise2, promise3])

// .then((posRes)=>{

// document.write(posRes);

// }, (errRes)=>{

// document.write(errRes);

// });

// Call Failed

// all method will execute only failed promise it will not highlight success

// Promise.race([promise1, promise2, promise3]).then((posRes)=>{

// document.write(posRes);

// }, (errRes)=>{

// document.write(errRes);

// });

// race method is used to know which promise will execute first

Promise.allSettled([promise1, promise2, promise3]).then((posRes)=>{

console.log(posRes);

}, (errRes)=>{

console.log(errRes);

});

// 0: {status: 'fulfilled', value: 'Hello\_1'}

// 1: {status: 'rejected', reason: 'Call Failed'}

// 2: {status: 'fulfilled', value: 'Hello\_3'}

// length: 3

</script>

</body>

</html>

1. what is promise

Ans: Promises are used to make Asynchronous calls

1. why promises

Ans: It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.

1. difference between callbacks and promises

Ans: A callback function is passed as an argument to another function whereas Promise is something that is achieved or completed in the future.

1. how to create promise

Ans: Promise() object is the predefined class, used to create the Promises

1. how to consume promise

Ans: We will consume Promise in two ways

1) then()

2) async & await

6) what is all() function

Ans: all() function is used to overcome the data redundancy (used to consume all functions at a time)

all() method will execute only failed promises, it will not highlight success.

7) what is race() function

Ans: race() method is used to know which promise will execute first.

8) what is allSettled()

Ans: allSettled() method will execute all success & failure promises.

9) what is async & await

Ans: "async and await make promises easier to write"

async makes a function return a Promise

await makes a function wait for a Promise

2023/04/26

<script>

let add=(num)=>{

return new Promise((resolve, reject)=>{

resolve(num+5);

})

}

let sub=(num)=>{

return new Promise((resolve, reject)=>{

resolve(num-3);

})

}

let mul=(num)=>{

return new Promise((resolve, reject)=>{

resolve(num\*2);

})

}

let div=(num)=>{

return new Promise((resolve, reject)=>{

resolve(num/2-2);

})

}

let consume=async ()=>{

let addRes=await add(5);

let subRes=await sub(addRes);

let mulRes=await mul(subRes);

let divRes=await div(mulRes);

document.writeln(addRes, subRes, mulRes, divRes);

}

consume();

//10 7 14 5

</script>

**Example:** ajax call

<!DOCTYPE html>

<html lang="en">

<head>

<title>Promises</title>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>

</head>

<body>

<script>

function restAPICall(){

return new Promise((resolve, reject)=>{

$.ajax({

method : "GET",

url: "https://www.w3schools.com/angular/customers.php",

success : (posRes)=>{

resolve(posRes);

},

error:(errRes)=>{

reject(errRes);

}

})

})

}

async function consume(){

let res=await restAPICall();

document.write(res);

}

consume();

//$ajax("https://www.w3schools.com/angular/customers.php")

</script>

</body>

</html>

2023/04/27

**Closures**

Function one accessing another Function data is called Closure

Ex:

function func\_one()

{

var x=100;

function func\_two()

{

Console.log(x);

}

}

func\_two accessing func\_one data is called Closure.

**Examplle:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Closure</title>

</head>

<body>

<script>

function func\_one(){

var x=100;

var y=200;

return ()=>{

console.log(x);

console.log(y);

}

}

console.dir (func\_one());

// Output:

// [[Scopes]]: Scopes[2]

// 0: Closure (func\_one)

// x: 100

// y: 200

</script>

<script>

// for(var i=0; i<5; i++)

// {

// //console.log(i);

// setTimeout(()=>{

// console.log(i);

// }, 5000);

// }

// 5 5 5 5 5

// for(let i=0; i<5; i++)

// {

// //console.log(i);

// setTimeout(()=>{

// console.log(i);

// }, 5000);

// }

// ES6

// 0 1 2 3 4

//ES9

//IIFE(Immidate Invokable Functional Expression)

for(let i=0; i<5; i++)

{

//console.log(i);

((i)=>{

setTimeout(()=>{

console.log(i);

}, 5000);

})(i)

}

// 0 1 2 3 4

// if any inner function is accessing outer function data, called as closure

// we can overcome closure with var keyword in 2 ways

// let (ES6)

// IIFE (ES9) Immediate Invokable Function Expression

</script>

</body>

</html>

**Class**

* Collection of variables and functions called as class.
* "new" keyword is used to create "object" to the class
* before "ES6" version, classes wont supported by JavaScript
* we can implement classes with the help of "constructor functions" before ES6 verion
* we can refer current class members with the help o 'this' keyword

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Constructor Function</title>

</head>

<body>

<!-- <script>

function class\_one(){

this.sub\_one="ReactJS";

this.sub\_two="NodeJS";

this.sub\_three="MongoDB";

}

let obj1=new class\_one();

document.write(obj1.sub\_one, obj1.sub\_two, obj1.sub\_three);

document.write("<br>");

let obj2=new class\_one();

document.write(obj2.sub\_one, obj2.sub\_two, obj2.sub\_three);

//ReactJSNodeJSMongoDB

//ReactJSNodeJSMongoDB

</script> -->

<!-- <script>

function class\_one(){

this.var\_one="Hello";

this.func\_one=function(){

return "Welcome";

}

}

let obj=new class\_one();

document.write(obj.var\_one, obj.func\_one());

//Hello Welcome

</script> -->

<!-- <script>

function class\_one(){

this.var\_one="Hello";

this.func\_one=function(){

return this.var\_one;

}

}

let obj =new class\_one();

document.write(obj.func\_one());

//Hello

</script> -->

<!-- <script>

function class\_one(){};

class\_one.prototype.var\_one="Hello Darling";

// prototype used to refer the "Constructor" function

let obj=new class\_one();

document.write(obj.var\_one);

//Hello Darling

</script> -->

<!-- <script>

function class\_one(){};

class\_one.prototype.var\_one="ReactJS";

class\_one.prototype.func\_one= function(){

return "NodeJS";

}

let obj=new class\_one()

document.write(obj.var\_one);

document.write(obj.func\_one());

//ReactJS NodeJS

</script> -->

<!-- <script>

function Parent(){};

Parent.prototype.var\_one="Parent Class";

function Child(){};

Child.prototype = Object.create(Parent.prototype);

</script> -->

<!-- <script>

function class\_one(){};

class\_one.prototype.var\_one="Parent Class";

function class\_two(){};

class\_two.prototype = Object.create(class\_one.prototype);

class\_two.prototype.var\_two="Child Class";

let obj=new class\_two();

document.write(obj.var\_one, obj.var\_two);

//Parent Class Child Class

</script> -->

<script>

function class\_one(){};

class\_one.prototype.var\_one="Hello\_1";

class\_one.prototype.func\_one= function(){

return "Hello\_2";

}

function class\_two(){};

class\_two.prototype =Object.create(class\_one.prototype);

class\_two.prototype.var\_two="Welcome\_1";

class\_two.prototype.func\_two= function(){

return "Welcome\_2"

}

let obj=new class\_two();

document.write(obj.var\_one, obj.var\_two, obj.func\_one(), obj.func\_two());

//Hello\_1 Welcome\_1 Hello\_2 Welcome\_2

</script>

</body>

</html>