**1.Closures**

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function's scope from an inner function.

/\* Local Variable \*/

function local() {

    var i = 2;//Local Variable

    console.log(i \* i);//2\*2=4

}

local();

/\*Console log define Outside\*/

/\*functoion loca(){

  /\*  var i=2;//Scope value Present Inside

}

/\*console.log(i\*i);//i is not definded

loca();

/\* global Variable \*/

var o = 4

function global() {

    console.log(o \* o)

}

global();

/\* Counter with Global Variable\*/

var counter = 0;//global Variable

function myfunction() {

    counter += 1//countr=counter+1//1 2 3 4

}

myfunction();//0+1=1

myfunction();//1+1=2

myfunction();//2+1=3

myfunction();//3+1=4

console.log(counter)//It takes local variable output so answer is 4.

/\*Counter with Global & local Variable\*/

var count = 0;

function myfunct() {

    var count = 0;//local variable

    count += 1

    console.log(count)//Its taekes Local Variable O/P=1 @ all the time

}

myfunct();

myfunct();

myfunct();

console.log(count)//Its Globle Variable so 0

/\*document.getElementById("p").innerHTML=counter;

/\* Using return \*/

function n() {

    var i = 0;

    i += 1;

    console.log(i)//1

    return i;//return function always return to the same function and perfrm same value//iNIALIZE

}

n()//1

n()//1

/\*ITS CRATE BUTTON IN THE PAGE \*/

/\*function on(){

    document.getElementById("p").innerHTML=n();

}

/\*nESTED fUNCTION IT INCREASE THE VALUE\*/

function y() {

    var f = 0;

    function k() {

        f += 1//f=f+1;

    }

    k();

    console.log(f)

    return f;

}

y()

function ao() {

    document.getElementById("p").innerHTML = y()

}

/\* To avoid all the nested and return type repetition use closure \*/

/\*Closure Use Self INVOING FUNCTION AND INSIDE SELF INVOING FUNTION  ANNOYNOMS FUNC\*/

var v = (function () {                                   //self Invoking Function

    var s = 0;

    console.log(s)//0

    return function () {                               //Annonymous Function Present Inside The Self Invoing Function

        s += 1;

        console.log(s)//1,2,3,4

        return s;

    }

}

)

    ();

function slf() {

    document.getElementById("p").innerHTML = v()

}

# 2.Hoisting

JavaScript **Hoisting** refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code.

Hoisting allows functions to be safely used in code before they are declared.

Variable and class declarations are also hoisted, so they too can be referenced before they are declared. Note that doing so can lead to unexpected errors, and is not generally recommended.

/\*Hoisting\*/

var a=10;

let b=20;

const c=13;

console.log(a)

console.log(b)

console.log(c)

/\* Hoisting Step\*/

var a;//Undefined

let h;//Unintialized

//const u;

a=1;

h=2;

u=1;

//hositing

//1 var & console

var f=12;//var f; f=12

console.log(f);//12

//2 Console & Var

console.log(d);//d is undefined

var d=12;

//3 var ,init & console

var f;

f=20;

console.log(f)//20

//4 var ,console,int

var ddd;

console.log(ddd)//undfind

ddd=32;

//1 let & console

let cc=23;

console.log(cc);//23

//2 console & let

//console.log(fff)//not intial

//let fff=13

//3 let inti console

let t;

t=43;

console.log(t);//43

//4 let,console,int

let hj;

console.log(hj)//undefined

hj=12

//5 ccc=12345;

//console.log(ccc)//not int

//let ccc;//Runtim errror

//1 const & console

const zzz=1234567;

console.log(zzz)//1234567

//2 cons & const

//console.log(vvvv)

//const vvvv=234;//not inti run time error

//3 const,init,cons

//const fffff;

//fffff='babu'

//console.log(fffff)//Unintialaized

//4  const,consol,ini

//const jj;//Unintialaized

//console.log(jj)

//jj=12222222;

//5 cons,con,in

//console.log(jjjjjj)

//cont jjjjjj;//unintialize

//jjjjjj=54

//Function Definition oly hoisting

//Function Dclerartion not hoisting

//Function CALL AT LAST

function v()

{

    console.log("hello")//hello

}

v();

//Function CALL AT first

vvv();

function vvv()

{

    console.log("hello")//hello

}

//FUnction dclaration

var xxxx=function(){

console.log("HI")

}

xxxx()

//xxxxx();

//var xxxxx=function(){

    //console.log("HI")//typeerror

  //  }

  //let

  let ccccc=function(){

    console.log("let")//let

  }

  ccccc();

  //let @1st

  //cccccc();

  //let cccccc=function(){

    //console.log("let")//error

  //}

  //Self Invoing Function

let nam="abc";

(function(){

    console.log(nam);

    let rr=78;

    console.log(rr)

})();

//2

let name="abc";

(function(){

    let name="ert";

    console.log(name);

    })();

//name

var dd={name:"Gowthami"}

console.log(dd.name);//Gowthami

console.log(ddd.name);//Undefined

var ddd={name:"Gowthami"}

//let

let cdd={name:"Gowmi"}

console.log(cdd.name);//Gowthami

console.log(dcdd.name);//Undefined

let dcdd={name:"Gowmi"}

3.Array Method

In JavaScript, Array is **a built-in global object that allows you to store multiple elements at once**.

var i=["apple", "Banana" ,"Cherry" ,"Dates"];

console.log(i);

console.log(i[0]);

console.log(i[1]);

console.log(i[2]);

console.log(i[3]);

console.log(i[4]);

i[2]="Jacfruit"

console.log(i)

var a=["aaaa","bbbb","cccc","ddddd"]

for(var j=0;j<=4;j++)

{

    console.log(a[j])

}

console.log(a.length);

const g=new Array('D','u','s','t');

//typeof

console.log(typeof(g))

console.log(typeof(a))

console.log(g)

//total value

console.log(g[0])

//array to string

console.log(a.toString('a'))

//shift

console.log(a.shift())//Removing 1st element of array

console.log(a)

//unshift

console.log(a.unshift('zzzzz'))//Adding new element to the 1st element of array

console.log(a)

//Array Method

var l=['D','h','a','n',]

var ll=['b','a','v','y','a']

var v=["aaaa","bbbb","cccc","ddddd"]

//Push

console.log(l)//Dhan

console.log(l.push('y' ,'a'));// add  ya

console.log(l)//Dhanya

//splice

console.log(v.splice('1','1',"Kiruba"))//remove 1

console.log(v)

//slice

console.log(v.slice('1','3',))//display 1 &2 kiruba,cccc

console.log(v.slice(2))//it display from 2,3

//pop

console.log(l)//Dhanya

console.log(l.pop('a'));//Delete a

console.log(l)//Dhany

//concat

console.log(l)//Dhany

console.log(ll)//bavya

console.log(l.concat(ll));// Dhanyabavya

//join

console.log(l.join(""))//Dhany

console.log(l.join(" "))//D h a n y

console.log(l.join("    "))//D  h  a  n  y

console.log(l.join(","))//D,h,a,n,y

//reverse

console.log(l.reverse( ))//ynahD

4.String Method

In JavaScript, **strings are used to represent and work with a sequence of characters**. A string can represent an object as well as the primitive data type. JavaScript automatically converts primitive strings to String objects so that it's possible to use String methods and access properties even for primitive strings.

var s1="Hello Everyone i am hre";

var s2="Hi are you there Hi";

//1.charAt()

console.log(s1.charAt(13))//e

//2.charCodeAt

console.log(s2.charCodeAt(1))//105

console.log(s1.charCodeAt(13))//101

//3.concat()

console.log(s1.concat(s2))//Hello EveryoneHi

console.log(s1.concat(s1))//Hello EveryoneHello Everyone

//4.endsWith() & startsWith()

console.log(s1.startsWith("H"))//true

console.log(s1.startsWith("l"))//false

console.log(s2.endsWith("i"))//true

console.log(s1.endsWith("i"))//false

//5.fromCharCode()

console.log(String.fromCharCode(101))//e

//6.includes()

console.log(s1.includes("Everyone"))//true

console.log(s1.includes("Everyones"))//false

//7.indexOf()

console.log(s1.indexOf("Hello"))//0

console.log(s2.indexOf("are"))//3

console.log(s2.indexOf("Hi"))//0

//8.lastIndexOf()

console.log(s1.lastIndexOf("hre"))//20

console.log(s2.lastIndexOf("Hi"))//17

//9.match()

var a="HI Everyone is doing GOOD hi"

console.log(a.match(/HI/g))//string['HI']

console.log(a.match(/is/g))//string['is']

console.log(a.match(/HI/i))//string['HI']

console.log(a.match(/is/i))//string['is']

console.log(a.match(/H/gi))//string['H','h']

//10.repeat()

console.log(a.repeat(3))//HI Everyone is doing GOOD hiHI Everyone is doing GOOD hiHI Everyone is doing GOOD hi

b="I LIKE TO WATCH TV most"

//11.replace

console.log(b.replace(/like/i,"like"))//I like TO WATCH TV

//12.search

console.log(b.search("TO"))//7

//13.slice()

console.log(b.slice(0,5))//I LIK

//14.split()

console.log(b.split(""))//(18) ['I', ' ', 'L', 'I', 'K', 'E', ' ', 'T', 'O', ' ', 'W', 'A', 'T', 'C', 'H', ' ', 'T', 'V']

console.log(b.split(" "))//['I', 'LIKE', 'TO', 'WATCH', 'TV']

console.log(b.split("  "))//['I LIKE TO WATCH TV']

//15.substr()

console.log(b.substr(1,7))//LIKE T//starts from 1 and dispay 7 charactrs

console.log(b.substr(-1))//V//revrs ordr

//16.substring()

console.log(b.substring(1,7))// LIKE  //

//17.toUpperCase

console.log(b.toUpperCase());//I LIKE TO WATCH TV MOST

//18.toLowerCase

console.log(b.toLowerCase());//i like to watch tv most

//19.trim

a="11111 rrrrr gggg"

b="wwsae"

console.log(b.padStart(9,"v"))//vvvvwwsae

console.log(b.padEnd(9,"v"))//wwsaevvvv

5.Sync and Async JS

**In synchronous operations** tasks are performed one at a time and only when one is completed, the following is unblocked. In other words, you need to wait for a task to finish to move to the next one. **In asynchronous operations**, on the other hand, you can move to another task before the previous one finishes.

//Sync

function mySync()

{

    var x=10;

    let y=20;

    const z=30;

    console.log("x value is = " +x)//10

    console.log("y value is = " +y)//20

    console.log("z value is = "+z)//30

}

mySync()

//Async

function myAsync()

{

    var xx=10;

    let yy=20;

    var zz;

    console.log("xx value is = " +xx)//10

    setTimeout(function(){

        console.log("This is set time out function");

         zz=30;

         console.log(zz)//30 async process completed and wor

    },5000

    )

    console.log("yy value is = " +yy)//20

    console.log("zz value is = "+zz)//undefind

}

myAsync()

6.Var,Let & Const

|  |  |  |
| --- | --- | --- |
| **var** | **let** | **const** |
| The scope of a *var*variable is functional scope. | The scope of a*let* variable is block scope. | The scope of a *const* variable is block scope. |
| It can be updated and re-declared into the scope. | It can be updated but cannot be re-declared into the scope. | It cannot be updated or re-declared into the scope. |
| It can be declared without initialization. | It can be declared without initialization. | It cannot be declared without initialization. |
| It can be accessed without initialization as its default value is “undefined”. | It cannot be accessed without initialization, as it returns an error. | It cannot be accessed without initialization, as it cannot be declared without initialization. |

//Variable ,let,constant

//Variable

console.log("var")

console.log("--------------")

//step 1 Decleration

var a;

a=10;

//2nd step Redeclaraion possible

var a=1;

//3rd step scope called inside  the bracket value

//var  are not scope specific

if(true){

    var a=99;

//reintialize possible

    a=65321;

    console.log(a)  //scope

}

console.log(a)

//functions scope

function vara(){

    var a=12;

    console.log(a)

}

vara()

//object deceration

var obj;

obj={name:'Kavuthmai S'}

//Redecleration

obj={name:'Kavuthami Selvaraj'}

console.log(obj)

console.log(obj.name)

//if with object

if(true){

    var obj={name:"SRSISSVM"}

    console.log(obj)

}

console.log(obj)

console.log(obj.name)

//function with object

function varab()

{

    var obj={name:"SRSISSVM"}

    console.log(obj)

}

varab()

console.log(obj)

console.log(obj.name)

//constant

console.log("const")

console.log("-----------------")

//step 1 Decleration

const b=12;

//2nd step Redeclaraion not possible

//const b=11;

//3rd step scope called inside  the bracket value

//const and let are scope specific

if(true){

   const b=111199;

//reintializen not possible

//b=65321;

   console.log(b)  //scope

}

console.log(b)

 //function scope

 function constb()

{

    const b=54;

    console.log(b)

}

constb()

//object deceration

const obje={name:'Nandhini S'}

//Redecleration

//const obje={name:'Nandhini Selvaraj'}

console.log(obje)

console.log(obje.name)

//if with object

if(true){

    const obje={name:"SRSI"}

    console.log(obje)

    console.log(obje.name)

}

console.log(obje)

console.log(obje.name)

//if with object

function constab()

{

    const obje={name:"SRSI"}

    console.log(obje)

    console.log(obje.name)

}

constab()

console.log(obje)

console.log(obje.name)

//let

console.log("let")

console.log("----------------")

//step 1 Decleration

let c;

c=13;

//2nd step Redeclaraion not possible

//let c=2211;

//3rd step scope called inside  the bracket value

//const and let are scope specific

if(true){

    let c=4444411119999;

  //reintialize possible

    c=65321;

  console.log(c)  //scope

 }

console.log(c)

//Redeclaraion not possisble

//c=32

//function scope

function letc()

{

    let c=1012;

    console.log(c)

}

letc()

//Object decleratiion

let objec;

objec={age:"13"}

//Redecleration not possible

//let objec={age:'24'}

console.log(objec)

console.log(objec.age)

//if with object

if(true){

let objec={age:"1"}

console.log(objec)

console.log(objec.age)

}

console.log(objec)

console.log(objec.age)

//if with object

function letab(){

    let objec={age:"1"}

    console.log(objec)

    console.log(objec.age)

    }

    letab()

    console.log(objec)

    console.log(objec.age)

7.Call Back & 8.Promise

Callbacks are functions that are passed inside the arguments of other functions, this procedure is valid in JavaScript because functions are objects and objects can be passed as arguments to functions

A [**Promise**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) is an object representing the eventual completion or failure of an asynchronous operation. Since most people are consumers of already-created promises, this guide will explain consumption of returned promises before explaining how to create them.

//waiting //pending  state

//resolve //fulfilled state

//reject //reject state

// const qp1 =new Promise((resolve,reject)=>{

  //  resolve("Success")

    //reject("failure")

//})

//const qp2 =new Promise((resolve,reject)=>{

  //  reject("failure")

//})

//qp1.then(console.log)//its goto success

//qp2.catch(console.log)//its goto filure

//const p1 =new Promise((resolve,reject)=>{

  //  setTimeout(()=>{

    //    resolve("Success 1")

    //},100)

//})

  //  const p2 =new Promise((resolve,reject)=>{

    //setTimeout(()=>{

      //  reject("failure 1")

    //},200)

    //})

//p1.then(console.log)//its display success @ 4sec

//p2.catch(console.log)//its display failure @5sec

//callbac hell

const f1=(val,callback)=>

{

    return callback(val)

}

const callback = (val)=>{

    return val +20;

    }

    console.log(f1(1,callback))

//promise callbac

const p1 = val => new Promise((resolve,reject)=>{

  setTimeout(()=>{

    resolve(val +20)

},1000)

})

p1(2).then(x => x+20).then(console.log)

//Promise all

const p1=new Promise((resolve,reject)=>{

    setTimeout(()=>{

        resolve("Success 1")

    },500);

})

const p2=new Promise((resolve,reject)=>{

    setTimeout(()=>{

        reject("failure 1")

    },1000);

})

//p1.then(console.log)

//p2.catch(console.log)

Promise.all([p1,p2]).then(console.log).catch(console.log)

//Promise Array

const pp1=new Promise((resolve,reject)=>{

    setTimeout(()=>{

        resolve("Success 2")

    },5000);

})

const pp2=new Promise((resolve,reject)=>{

    setTimeout(()=>{

        resolve("Success 2.1")

//reject("failure")

    },5000);

})

//p1.then(console.log)

//p2.catch(console.log)

Promise.all([pp1,pp2]).then(console.log).catch(console.log)

//Promise Race

const qpp1=new Promise((resolve,reject)=>{

    setTimeout(()=>{

        resolve("Success 3")

    },1000);

})

const qpp2=new Promise((resolve,reject)=>{

    setTimeout(()=>{

reject("failure 3")

    },200);

})

//p1.then(console.log)

//p2.catch(console.log)

Promise.race([qpp1,qpp2]).then(console.log).catch(console.log)

9.Async and await

**An async function is a function declared with the async keyword, and the await keyword is permitted within it**. The async and await keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains

async function myfun(){

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

    resolve("Succss 1")

 },3000))

}

async function await(){

    const output = await myfun();

    console.log(output)

}

await()

10.Classes

In JavaScript, a class is **a type of function and are declared with the class keyword**. You need to use function expression syntax to initialize a function and class expression syntax to initialize a class.

11.What is this () and super ()?

super() acts as immediate parent class constructor and should be first line in child class constructor. this() acts as current class constructor and can be used in parametrized constructors. Override. When invoking a superclass version of an overridden method the super keyword is used

//class entity name(student)

// class innstanceat time to pass value

//constructor map the class details

//this keyword working

class Student {

    constructor(name,age,mobile,roll)

    {

this.cl\_name=name;

this.cl\_age=age;

this.cl\_mobile=mobile;

this.cl\_roll=roll;

    }

}

user1 = new Student("AAAA","10","9876543321","32145");

user2 = new Student("BBB","12","987654331","43245");

user3 = new Student("AAAA","10","9876543321","32145");

user4 = new Student("AAAA","10","9876543321","32145");

console.log(user1.cl\_name);

console.log(user1.cl\_age);

console.log(user1.cl\_mobile);

console.log(user1.cl\_roll);

console.log(user2.cl\_name);

console.log(user2.cl\_age);

console.log(user2.cl\_mobile);

console.log(user2.cl\_roll);

//super keyword

//School Super Class

//Student Sub Class

class School extends Student{

    constructor(name,student\_name,         student\_age,

        student\_mobile,student\_roll)

    {

    super(student\_name,         student\_age,

        student\_mobile,student\_roll);

        this.school\_name=name;

    }

    print(){

        return this.cl\_name + " studies in this" +   this.school\_name + "school";

    }

}

school1 = new School("SRSI","Cheran","10","987654321","01");

school2 = new School("SSVM","Yash","10","978643215","30");

console.log(school1.print());

console.log(school2.print());

Object:

an object is **an unordered collection of key-value pairs**. Each key-value pair is called a property. The key of a property can be a string. And the value of a property can be any value, e.g., a string, a number, an array, and even a function.

let person = {

    name:   "Kavuthami S",

    age:    23,

    isAlive: true,

    gender: "female",

    address: "Coimbator,TN",

    //adding family details

    family: {

        fathername: "Selvaraj K",

        mothername: "Rajammmal D",

        sistername: "Nandhini S",

        husbandname: "Kirubanantham C"

    }

}

//Dot notation

console.log(person)

console.log(person.name)

console.log(person.age)

console.log(person.isAlive)

console.log(person.gender)

console.log(person.address)

console.log(person.family)

console.log(person.family.fathername)

console.log(person.family.mothername)

console.log(person.family.sistername)

console.log(person.family.husbandname)

//Bracet notation

console.log(person['name'])

console.log(person['isAlive'])

console.log(person['gender'])

console.log(person['age'])

console.log("SUB OBJECT")

console.log(person.family['fathername'])

console.log(person.family['mothername'])

console.log(person.family['sistername'])

console.log(person.family['husbandname'])

Inheritance

Inheritance **enables you to define a class that takes all the functionality from a parent class and allows you to add more**. Using class inheritance, a class can inherit all the methods and properties of another class. Inheritance is a useful feature that allows code reusability.

class family{

    constructor(myname)

    {

        this.familyname=myname;

    }

    a()

    {

        return "My name is "+this.familyname;

    }

}

let x=new family("Kavuthami S");

console.log(x.a());

class husband extends family{

    constructor(myname,husband){

        super(myname);

        this.husbandname=husband;

    }

    h()

    {

        return this.a() + " and My husband name is "+this.husbandname;

    }

}

let y=new husband("Kavuthami","Kiruba");

console.log(y.h());

//Getters and setters

12.Ternary Operator

//Ternary Operator

//voting

let age=15;

let t = age >= 18 ? "Eigible to vote":"Not Eligible"

console.log(t);

var x=20;

let y= x >= 18 ? "Eligible" : "Not Eigible"

console.log(y);