# Javascript

## Closure

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. In JavaScript, closures are created every time a function is created, at function creation time.

//Closure Example

function makeFunc() {

const name = 'Geethaa';

function displayName() {

console.log(name);

}

displayName();

}

makeFunc();

## Hosting

Hoisting is the default behavior of moving all the declarations at the top of the scope before code execution.

JavaScript allocates memory for all variables and functions defined in the program before execution. Also, in Javascript is first declaring the variable and then initializing them.

**Declaration –> Initialisation/Assignment –> Usage**

For Example:

// Variable lifecycle

let a; // Declaration

a = 100; // Assignment

console.log(a); // Usage

// hoisting

function Hoist(){

a = 10;

var b = 50;

}

Hoist();

console.log(a); // 10

console.log(b); // Error : b is not defined

# 

//concat

var a = ["Everything", "is", "possible"]

var b = ["when", "you", "focus"]

console.log(a.concat(b))//Output :[ 'Everything', 'is', 'possible', 'when', 'you', 'focus' ]

## ToString

The tostring method will change the typeof array to tostring.

For Example: In the below code, we can see before changing it to tostring the typeof variable(num) is “number” but after the tostring methods applied, the typeof the variable(str) is “String”.

//tostring

var num = 23;

console.log(typeof (num))//typeof num before changing to string is Number

console.log(num.toString())//output 23

console.log(typeof (num.toString()))// typeof num after changed tostring is string

## Push

Push method will help us to add or concat any word or string at the end and return as a single string.

For Example: In the below code, we can see that after using the push method “Everthing is” changed to “Everything is Possible” , by concating the parameter which passed.

//push

var pusmethod = ["Everything", "is"]

pusmethod.push("possible")//adding any data in the last with the help of push

console.log(pusmethod)//[ 'Everything', 'is', 'possible' ]

## Shift

Shift method will remove the first letter or index of the array and return the removed string as an output.

For example:In the below code, we can see the output was returned as “Trying” and once after the shift method is applied the array will give without the first array.

//shift

var shiftmethod = ["Trying", "out", "shift", "method"]

console.log(shiftmethod.shift())//output Trying

console.log(shiftmethod)//after removing the first element it will give [ 'out', 'shift', 'method' ]

## Unshift

Unshift method return add one or more elements in the beginning of the array and returns the new array

For Example: In the below code, we can see the parameter passed as “JavaScript:'' was concatenated in the beginning of the array and returned as a single new array.

//Unshift

var Unshiftmethod = ["Trying", "out", "Unshift", "method"]

Unshiftmethod.unshift("Javascript:")

console.log(Unshiftmethod)//[ 'Javascript:', 'Trying', 'out', 'Unshift', 'method' ]

## Reverse

Reverse method will reverse the array, the first element will be the last element and the last element will be in first.

For example: In the below code, we can see the array has been reversed

//reverse

var revmethod = ["Trying", "out", "reverse", "method"]

revmethod.reverse()

console.log(revmethod)//output :[ 'method', 'reverse', 'out', 'Trying' ]

## Length

Length method will return the number of elements in the given array sets.

For example: In the below code, we can see the output is returned as 4 which the elements count in the array.

//length

var revmethod = ["Trying", "out", "length", "method"]

console.log(revmethod.length);//output 4

## Find

Find method will return the value of the first element in the array which passes the test case or condition.

For Example:In the below code, for the condition <30 gets true in the second index so it returns output as 23.

//find

var fndmeth = [54, 23, 84, 20]

//find returns the value of the first element in an array that passes a test .

function greaternum(findmeth) {

return findmeth < 30

}

console.log(fndmeth.find(greaternum))//output :23

## FindIndexof

The FindIndexof method will help us to find the index of any element in the array set.

For Example: In the below code, we can see the index of 54 is “1” so t returned output as “1”

//findIndexof

var fndIndexofmeth = [23, 54, 84, 20]

console.log(fndIndexofmeth.indexOf(54))//1

## Filter

Filter will check the array with the help of the condition given and then return the first element which passes the condition/test case.

For Example: In the below code, the condition was marks >= 35 and it returns 56 because in the array 56 is the first element which passes the condition so it returns 56.

//filter

var marks = [34, 56, 78, 56, 90, 48, 35]

function checkmarks(marks) {

return marks >= 35

}

console.log(marks.find(checkmarks))//output 56

## Map

Map method will create a new array with the results of the condition given.

For example: In the below code, the condition was square root of level and it returns the output in an array with the results.

//map

var level = [4, 9, 16, 25, 36]

function methodmap(level) {

return Math.sqrt(level)

}

console.log(level.map(methodmap))//output: [ 2, 3, 4, 5, 6 ]

## Slice

Slice method will return a portion of an array with the help of start and end (end will not be included).

For example: In the below code, we can see the output was returned from the index value of 0 to 2 which is not included in the index 3.

//slice

var methslice = ["Trying", "out", "slice", "method"]

function basic(methslice) {

return methslice.slice(0, 3)

}

console.log(basic(methslice)) //output: [ 'Trying', 'out', 'slice' ]

## Splice

Splice method will replace the portion of an array with values using the start and end array index(end will not be included).

For example: In the below code, we can see the output was returned by changing new values from the index value of 1 to 2.

//splice

var splicemeth = ["Trying", "out", "slice", "method"]

splicemeth.splice(1, 2, "the", "splice")

console.log(splicemeth) //output : [ 'Trying', 'the', 'splice', 'method' ]

## Sort

Sort method will arrange the array in an ascending or alphabetical order of the array.

For example: In the below code, we can see the numbers in variables are arranged in ascending order in output.

//sort

var srtmethod = [1, 6, 8, 5, 3, 2, 9, 4, 7]

srtmethod.sort()//will sort in ascending or alphabetical order.

console.log(srtmethod)//output :[1, 2, 3, 4, 5,6, 7, 8, 9]

## Reduce

The reduce method will check element by element, at each step adding the current array value to the result from the previous step.

For Example:In the below code, we can see that methreduce was added completely and returns a single string.

//reduce

var methreduce = [35, 24, 21, 80]

function getsum(value, num) {

return value + num

}

console.log(methreduce.reduce(getsum, 0))// output :160

## Every

Every method will check if all the elements pass the condition, if yes then it will return the true. If one element check is failed as per condition it will return false.

For Example: In the below array we can see the test case will fail for the first element itself so it returns output as false.

//every

var methevery = [34, 90, 14, 18, 90, 45]

function ages(methevery) {

return ages >= 18

}

console.log(methevery.every(ages))//every checks all the elements in array is greater than or equal to 18 if every value in array passes the condition it will return the true else return false

## Some

The some method checks the elements in the array and verifies at least one element in the array passes the test, if the condition is true for any one of the elements then the output will be true. Else it will return false.

For Example: In the below code, we can see the condition is true for one or two elements hence it returns true as output.

//some

var methsome = [8, 9, 10, 25, 24, 13]

function isage(methsome) {

return methsome < 10

}

console.log(methsome.some(isage))//output : true

## Includes

Includes method will verify whether an array includes any particular value, if yes it will return true else it will return false.

For Example: In the below code, we can see the state “TamilNadu” was included in the array so the output is true

//includes

var state = ["TamilNadu", "Karnataka", "Andhara", "Maharastra", "Delhi"]

function country(state) {

return state

}

console.log(state.includes("TamilNadu"))//output :true

## 

## IndexOf

The method index of will return the index value of the array.

For Example: In the below code, we can see the index value of the word “is” is “1” so it returns output as “1”.

//indexof

var array = ["All", "is", "well"]

console.log(array.indexOf("is"))//output :1

# **Variables**

Variables is a container which store the data or value, which will be string, Array or any other values.

For Example:In below code, we will store the var x=13 and we will return it .

//Variables

var x=13

var y=22

console.log(x)//output :13

# **Let**

Variables can’t be redeclared in the concept of let.

For example : We can see in the below code, for the first example it will return an error message. But in second example it will return the output.

Example 1:

//LET

let a=45;

let a=56;

console.log(a)//with the help of let we cannot redeclare the value

//output :dentifier 'a' has already been declared

Example 2:

let x=45;

let z=56;

console.log(x)//output : 45 now it can give the output which is not redeclared.

# **Constant**

Constant cannot be redeclared and reassigned. It is like a constant value which is assigned throughout the end. You can’t change.

For example : In the below code, we can see the constant variable is redeclared so it will throw a error.

Example 1:

//constant

const xy=["Geethaa"]

xy=["shobana"]// output :Assignment to constant variable.

console.log(xy)

Example 2:

//constant

const xy;

xy=87;//this will also throw an error

# **Synchronous**

Synchronous means to be sequence which is like the code will execute one by one /line by line.

For Exmaple: In the below code, we can see the first output was “Geethaa” from the first line and the next it will print for “Shobana”.

function sync(){

console.log("Geethaa")

console.log("Shobana")

}

sync()

//output will be printed as

//Geethaa

//Shobana

//because here the code will be working one by one only.

# Asynchronous

Asynchronous code will execute the program immediately without waiting for the current one to be executed.

For Example: In the below code, we can see the output will be First and Third will be printed without waiting for the execution of the second function.

//Asynchronous

console.log("First")

setTimeout(() =>{

console.log("Second")

},3000);

console.log("Third")

//output

//First

//Third

//after 3 seconds the "Second" will print

# Callback

Callback is a function passed into another function as an parameter then invoked by the other function.

For example :In the below code we can see the callback function is called in the sum function as a parameter and it got executed invoked by other function. We can see that the sum function was called and it printed the **Calaculation happening** and then it goes to the argument function (callback) and gave the output as **Total is =20**

//Callback

num1=10

num2=10

function callback(){

num=num1+num2

console.log("Total is=" + num)

}

function sum(num){

console.log("Calculation happening")

num();

}

sum(callback)

//output

//Calculation happening

//Total is=20

# **Promise**

The promise is one these states pending , neither successful or failed

In promise we will use the future cases then which gives after function what needs to be done neither success or failure.

For example: In the below code, we can see if that neither resolve or reject will happen when the function is true or false.

//Promise

var p1 = new Promise(function (resolve, reject) {

num = 1

if (num > 5) {

resolve(num)

} else

reject(num)

})

p1.then(function (resolve) {

console.log("Ok")

})

.catch(function(reject) {

console.log("Fail")

})

//if we give num=1 u will geth Fail as output

//if we give num=6 u will geth OK as output

Syntax

In syntax we can see the producing code and the Consuming code, in this promis object obtains both the producing code and calls the Consuming code. When producing code obtains result it should call one of the two callbacks. Like wise we have many methods in promise.

let myPromise = new Promise(function(myResolve, myReject) {

// "Producing Code" (May take some time)

myResolve(); // when successful

myReject(); // when error

});

// "Consuming Code" (Must wait for a fulfilled Promise)

myPromise.then(

function(value) { /\* code if successful \*/ },

function(error) { /\* code if some error \*/ }

# 

# **Async and Await**

In the Asynchronous we have already seen that it will not wait for function to execute it will give the next result but here with the help of promise we can make the async to wait and the execute the result. This is called as Async.

An async function without an await expression will run synchronously.If there is an await expression inside the function body, however, the async function will always complete asynchronously.

For example:The async with await and promise is equivalent

async function foo() {

await 1

}

For Example: We can see the first output is total 22 which is displayed after 4 secs

and then “ok” is printed

//Async

var p1 = new Promise(function (resolve, reject) {

setTimeout(() => {

num1 = 12,

num2 = 10

num = num1 + num2;

console.log("Total is " + num)

resolve()

}, 4000)

})

p1.then(function (resolve) {

console.log("Ok")

})

//output (first wait for 4 secs and the it will print ) "Total is 22"

//then print "Ok"

# **Class, Objects and this:**

Classes are a template for creating objects. One way create a class is with the class declaration and with the constructor. Constructor is a special method which creating and initializing the an object created with class.

In this below code we can see class has been created and with the help of constructor we are initailizing the objects. And this is determined by how a function is called.

class monthlyNeeds{

constructor(vegetables,snacks,groceries){

this.vegetables=vegetables;

this.snacks=snacks;

this.groceries=groceries;

}

view(){

return this.vegetables+","+this.groceries+","+this.snacks;

}

}

var obj=new monthlyNeeds("Tomatoes","Biscuts","toothpaste")

console.log(obj.view())

# Inheritance and super:

Inheritance happens with the help of extends keyword, when we need to extend the details from the parent class to chile class. The super keyword calls the parent class's constructor and binds the parent class's public fields, after which the derived class's constructor can further access and modify this.

In the below code we can see the child class extends monthlyNeeds and with the help of super keyword we binded the parent class constructors. We can also extend the main/parent class for many other child classes.

class monthlyNeeds {

constructor(vegetables, snacks, groceries) {

this.vegetables = vegetables;

this.snacks = snacks;

this.groceries = groceries;

}

view() {

return this.vegetables + "," + this.groceries + "," + this.snacks;

}

}

var obj = new monthlyNeeds("Tomatoes", "Biscuts", "toothpaste")

console.log(obj.view())

class budget extends monthlyNeeds {

constructor(vegetables, snacks, groceries, total) {

super(vegetables, snacks, groceries)

this.total = total

}

amount() {

return this.vegetables + "," + this.groceries + "," + this.snacks + "," + this.total

}

}

var obj1 = new budget("Tomatoes", "Biscuts", "toothpaste", "4000")

console.log(obj1.amount())

//output for first console.log Tomatoes,toothpaste,Biscuts

// output for second console.log Tomatoes,toothpaste,Biscuts,4000

~THE END~