1. **Closure**

1. Accessing outer variable to inner function.

2. Accessing global scope

3. must return values in closure.

function a1(){

var sd = 10;

console.log(sd+ "out")

function a2()

{

 return sd++;

}

return a2();

​

}

​

var aw= a1();

aw;

1. **Callback function**
2. Callback function is another function action completed after that current action completed.

2. Callback function we are passing as a function name as a parameter.

**Example:**

function add(a, fun)

{  
 fun();  
 console.log(“second function completed after first function “ + a);  
}

function add1()

{alert(“function two”);}

add(“prabu”, add1);

1. **Scope**

**Block scope** = only let in block scope also if ,loops, switch;  
**function scope** = separate function inside access only   
**Global scope** = can access all places

1. **Hoisting**

Only take to top of declarations is called hoisting.

Example:

a = 10; // **Initialization**

console.log(a);

var a; // **Declarations**

1. **Call, Apply, Bind**

**Why we are using call apply bind:**

* We want access the object methods in function so we are using call,apply,bind.
* We are creating one link to object and function

**Bind** = this will give new function and also bind object and function

**Example :**var a= {

name: "prabu",

getname : function()

{

return this.name;

}

};

function add()

{

console.log (this.getname());

}

Var t = add.bind(a);

**Call**

1. call method combines **obj+ function give new result**
2. Also pass parameters in call method **string**

Example:

**Apply**

1. Passing parameters in **array.**

**Example of call method**

var a= {

name: "prabu",

getname : function()

{

return this.name;

}

};

function add(a)

{

console.log (this.getname()+ a);

}

add. call(a,"mani");

**Example of Apply method**

var a= {

name: "prabu",

getname : function()

{

return this.name;

}

};

function add(a)

{

console.log (this.getname()+ a);

}

**add. apply(a,["mani"]);**

//Array is an Parameter

**6.Bubbling Vs capturing**

Bubbling= default **false** bubbling

Capturing = when you adding third parameter **true** is capturing.

**Html**

<div id="parent">Parent

<div id="child" onclick="event.stopPropagation()">Child</div>

</div>

**<script>**

var parent = document.querySelector('#parent');

parent.addEventListener('click', function(){

console.log("Parent clicked");

});

var child = document.querySelector('#child');

child.addEventListener('click', function(){

console.log("Child clicked");

});

</script>

**Action:**

* When we click the child it will bubbling on parent so this called bubbling.
* You can stop child function inside **stopPropagation()** method.
* We want stop current exection suddenly use **stopImmediatePropagation().**

**event. stopPropagation()**

It will stop the bubbling . Example bottom to top stop top.

**event. preventDefault()**

It will stop the default behavior action.

Example submit button using when use **preventDefault it will stop the action.**

**event.stopImmediatePropagation()** this will stop immediately after completed current function.

**7. SetTimeot**   
 Executes the particular event after time interval.  
  
**Example**:

SetTimeot (function(){

alert("s");

},1000);

**SetIntervel**

Exexutes the every time again and again that interval time function.

**Example**:

setIntervel (function(){

alert("s");

},1000);

**Removing event Listeners:**

**clearTimeout(myVar);**

**clearInterval(myVar);**

**Var** myVar;

function myFunction() {  
  myVar = setTimeout(function(){ alert("Hello"); }, 3000);  
}  
  
function myStopFunction() {  
  clearTimeout(myVar);  
}

**8. addEventListener Vs event.delegation**

The addeventlistener is an add the event after page load have already existing dom elements.

**Drawbacks:**

Cannot fire event after page load added new events.

**Example:**

**Html:**

<ul id=”v1”>

<li>1</li>

<li>2</li>

<li>3</li>

</uI>

**Script:**

Var lis= document.getElementById(“v1”);

**Lis.addEventlistener(“click”, fun1, bubbling or capturing);**

**Remove event listener methods**

Element. **removeEventListener**(“click”,fun1, **bubbling or capturing**);

**event.delegation**

Event delegation is and solves the addeventlistener drawbacks. If you are adding

Parent to the event listener then child what are parents say it will react same. So

New elements added also it will work properly.

**9. Objects**

1.Objects are consists of key and value pairs.

2. object have the following properties

1. writable

i) You can modify the values

2.enumerable

i) It can be iterate the values

3.configurable

i) You can set non-writable

ii) you can set non-enumerable

iii) you can delete use delete keyword

**object creating methods**

1. Literal method:

Example:

Var a= {

};

1. Var x= Object.create();
2. **Object Constructor**

Var x= new Object();

1. **Constructor function**

Using NEW keyword creating objects

function Person(id)

{

This.id= id;

}

Var x= new Person(23);

**Accessing Objects Properties Methods:**

Var obj = {

a:10,

a name:30;

};

1. **Dot notation**

Using dot notation we can access=> [Obj.a]

1. **Bracket Notation**

Using array bracket we can access =>Obj[“a”]

1. **Space accessing following syntax**

Obj[“a name”]

**Delete property**

Using delete keyword we can delete the properties.

**Example:**

delete obj.a;

**Iterate ways of methods:**

1. **For in method**

for(var x in obj)

{

Obj(x);

}

1. **For loop[ when we are using array objects that time only]**

For loop using method

for(var i=0;i<obj.length;i++)

{

}

**Array Objects creating methods:**

**Method:1**

var ae= [

{index: [{name:"Prabu",age:10},{name:"mani",age:30}]},

{index: [{name:"Prabu",age:10},{name:"mani",age:30}]},

];

**Method:2**

Var ae = [

{ name:"Prabu11",age:10},{name:"mani11",age:30},

{ name:"Prabu1",age:10},{name:"mani1",age:30}

];

**Clone methods objects:**

1. **Shallow copy methods [changes happen in original object]**
2. Its just copy the memory Address only. No values.

Var obj1 = {

a:10;

};

Var obj2={

};

Obj2= obj1// **shallow copy**

**Shallow copy method2:**

Var newObj= Object.assign({},obj);

<https://scotch.io/bar-talk/copying-objects-in-javascript#toc-the-naive-way-of-copying-objects>

1. **Deep Copying objects**

The deep copy is an copy the original object and its have an original reference not copying Address.

**Var newObj = JSON.parse(JSON.stringfy(obj));**

**Using for loop**

**Using for loop we can perform deep copy**

**Spread Opertor**

**Same like deep copy**

Var obj1= {…obj};

**Object Methods:**

1. **Object.assign({}, Obj)**

Copy the values one obj to another obj.

1. **Object.create()**

Creates an new object.

1. **Object.defineProperty()**

Adds the only one property and replace the values in existing property values.

**Example:**

Object.defineProperty(object1, 'property1', {

value: 42,

writable: false

});

1. **Object.defineProperties()**

Adds the multiple property and replace the values in existing property values.

**Example:**

Object.defineProperties(object1, {

property1: {

value: 42,

writable: true

},

property2: {}

});

1. **Object.keys() Vs Object.getOwnPropertyNames()**

It will return keys on array.

**Example:**

Var ad= {

Name:”mani”,

Id:10

};

Object.**keys**(ad) **=> [“Name”,”Id”];**

1. **Object.values()**

It will return values on array.

**Example:**

Var ad= {

Name:”mani”,

Id:10

};

Object. **values** (ad) **=> [“Mani”,10];**

1. **Object.entries()**

It will return key and values on array on each item separately.

**Example:**

Var ad= {

name1:” mani”,

name2: “mani1”,

};

Object. **entries** (ad) **=>** ["name1", "mani"],["name2", "mani1"],

1. **Object.freeze()**

Once freeze the obj we cant delete and change properties values.

1. **Object.seal()**

We cannot add new properties and we can modify existing properties

Values.

We cannot delete the object values.

1. **Object.isSealed()**

Check whether object is sealed or not.

**Array**

Array is storing multiple data type values. Like string, Boolean, number,Object.

**Array creating ways**

**Method1:**

Var arr= [];

**Method2:**

Var arr1= new Array();

**Find length of array**

Array.length;

**Loop Array elements**

1. **ForEach**

Its will iterate all array elements.

**Example:**

var arr1= [1,3,7,72];

arr1.forEach(function(value){

alert(value);

}

);

1. **for of loop**

Its will iterate all array elements.  
when we are using for of loop we need give **index-1** value that time only proper values

**Example:**

var a1= [1,2,3,4];

for(var i of a1)

{

console.log(i+ "index"+ a1[i-1]+ "value");

}

1. **for loop**

we know that for loop iteration.

**Cloning Array methods**

**Shallow Copy:**

**method: 1**

It only copy the memory address:

var arr1= [1,2,3];  
 var newArray = arr1;

**method:2**

we can copy using splice method.

var ar2= [1,2,3,4];

var a1= ar2.splice(0);

**a1// Have the all original array but it will change original array.**

**Deep Copy in array methods:  
   
 Method1:**

**Using slice method we can do this**

var d= [1,2,3];

var new1= d.slice(0,length-1);

**method 2:**

using for loop

**Method:3**

**Var newObj = JSON.parse(JSON.stringfy(arr));**

**Method:4**

Using for loop

Using for loop we can perform deep copy

**Method:5**

Spread Opertor

Same like deep copy

Var arr= […obj];

**Array Methods**

The following methods of array:

1. concat

join two arrays.

var hege = ["Cecilie", "Lone"];  
var stale = ["Emil", "Tobias", "Linus"];  
var children = hege.concat(stale);

//Cecilie,Lone,Emil,Tobias,Linus

1. entries()

return key and value on array.

[“0”, “Prabu”] 🡺 you want to take this values **x.next().value;**

1. valueOf()

Return values on array.

1. every()

Checks the all condition where getting false that place stop remaining not executed.  
Return false Boolean values.

1. Filter

It passes all condition getting result it will create new array.

Return values in new array.

1. Find

Check the condition when get first true that values return the values.

Return the first true condition value.

1. Some()

When get true it will stop.

And return Boolean value.

1. Findindex

It will return the first true index values.

Otherwise Return -1.

1. ForEach()

For array iterate purpose.

1. Array.from

We can copy the values one array to another array.

Also we can covert string values to array.

1. indexOf()

find the first occurrence return the index values.

Not find return -1.

1. lastIndexOf()

find the last occurrence to first and return the index values.

Not find return -1.

1. Pop()

Removes the last element from an array.

1. PUSH()

Adds the new element at the last of an array.

1. unshift()

Adds the new element at the first of an array.

1. shift()

Removes the first element from an array.

1. isArray()

check wheather array or not.

18. join()

Join the array values into the string.

Return string values separted by comma.

19.Map()

Map method create a new array

20) reduce

This method gives the single value.

Left to right value.

21) reduceRight

This method gives the single value.

Right to left value.

21) reverse()

reverse the arrays left to right and right to left.

22) sort()

Sort the values in alphabalts.

**23) splice()**

Splice method changes the original array.

Also give only removed item in array.

Also we add new elements in array.

**Example:**

var ss= [1,2,3,4,5];

var s1= ss.splice(1,1);

**Ans:** S1= [2];

**24) slice**

Slice method give the selected elements values.

It cannot remove values from original array.

**Example:**

var sf= [1,2,3,4,5];

sf= sf.slice(2,4);

sf= [3,4]

var ages = [32, 33, 16, 40];

function checkAdult(age) {

return age >= 18;

}

ages.filter(checkAdult);

(3) [32, 33, 40]

var ages = [32, 33, 16, 40];

function checkAdult(age) {

return age >= 18;

}

ages.every(checkAdult);

false

var ages = [32, 33, 16, 40];

function checkAdult(age) {

return age >= 18;

}

ages.find(checkAdult);

32

Filter = all values pass and pass condition only return array. False condition omitted

Every= once get false it will stop the execution and return false

Some= first true condition return true stop execution

Find= first true condition return values stop execution

FindIndex = this method return the true condition index value.

**String creating methods**

**Method1:**

Var str= new String();

**Method2:**

Var str= “”;

**String Accessing methods**

Str[]== same like object

**Cloneing strings**

**Deep Cloneing:**

**Method1:**

var str1= "prabu s";

var news= JSON.parse(JSON.stringify(str1));

**Method2:**

var str1= "prabu s";

var news= str1.slice(0,end);

**method3:**

var str1= “Prabu 1”;

str1.split(“”);

use join method join string.

**Method 4:**

var str1= “Prabu 1”;

str1.split(“”);

using for loop join string.

**Array Methods**

1. **CharAt=** Returns the character at the specified index.
2. **Concat() = join two strings**
3. **indexOf == gives the first search index value.**
4. **lastIndexOf== gives the last to find first index value.**
5. **Match()== gives the matching results like regular**

**Expression.**

1. **search()== gives the matching results like regular**

**Expression.**

1. **Slice = extracts strings purpose**

**Not change original array.**

1. **substr(start,number of char)**

**it will two parametes. Start and how many elements.**

1. **Split()**

**Split method splits the strings in string.**

1. **toString()**

**convert to string format.**

1. **toUpperCase() And toLowerCase()**

**converts to uppercase text**

**lowercase text.**

1. **Trim()**

**Remove the empty space.**

**OOPS CONCEPTS IN JS**

<http://www.w3processing.com/index.php?subMenuLoad=javascript/Function/FunctionEncapsulation.php>

<https://www.javatpoint.com/javascript-oops-polymorphism>

<http://www.w3processing.com/index.php?subMenuLoad=javascript/Function/FunctionPolymorphism.php>

**Javascript inheritance**

Inheritance method is **used to inherit the parent class.**

Also Access the parameters.

**Parent**

function Person(first, last, age, gender, interests) {

this.name = {

first,

last

};

this.age = age;

this.gender = gender;

this.interests = interests;

};

**Child**

function Teacher(first, last, age, gender, interests, subject) {

**Accessing Parent class properties**

console.log(**Person.apply(this, [first, last, age, gender, interests]));**

this.subject = subject;

}**Accessing Parent class methods**

**Teacher.prototype= Object.create(Person.prototype);**

var t1 = new Teacher(1,2,3,4,56,7);

**Inheritance Example:**

function Person(first,last)

{

this.name1 =first;

this.name2= last;

}

**function Person1(first,last,age)**

{

Person.call(this,first,last);

this.age= age;

}

Person1.prototype.get1= function()

{

console.log(this.name1+ this.name2 + this.age);

}

**var x1= new Person1("mani1","Raj1",33);**

**console.log(x1.get1());**

**Javascript Encapsulation**

Encapsulation is method of consists of **data and methods in a single unit.**

Encapsulation is an method hiding the data and keep private data.

**Keys:**

Using Encapsulation we cannot access properties directly and we can store data into public Methods and we can access.

We make data in var keyword in private.

**Javascript Polymerphism**

<https://stackoverflow.com/questions/27642239/what-is-polymorphism-in-javascript>

Polymerphism is a take to one method and overwrite the as per the required.

Polymerphism is a reusability.

Only inherits needed property from parent class remaining thing override in a poly.

function Person(age, weight) {

this.age=age;

this.weight=weight;

this.getInfo=function() {

return "I am " + this.age + " years old " +

"and weighs " + this.weight +" kilo.";

}

}

function Employee(age, weight, salary){

Person.call(this,age, weight);

this.weight=weight;

this.getInfo=function() {

return "I am " + this.age + " years old " +

"and weighs " + this.weight +" kilo " +

"and earns " + this.salary + " dollar.";

}

}

Employee.prototype= new Person();

Employee.prototype.constructor=Employee;

// The argument, 'obj', can be of any kind

// which method, getInfo(), to be executed depend on the object

// that 'obj' refer to.

function showInfo(obj) {

document.write(obj.getInfo()+"<br>");

}

var person = new Person(50,90);

var employee = new Employee(43,80,50000);

showInfo(person);

showInfo(employee);

**Javascript Asynchronous and synchronous**

Javascript is always Aschronous

**Asynchronous**

Asynchronous request **Not blocks the client until operation completes**

**Page also not refresh.**

**synchronous**

synchronous request blocks the client until operation completes

**AJAX in js**

<https://www.javatpoint.com/ajax-tutorial>

**Ajax is** Asynchronous operation.

Asynchronous JavaScript and XML

Xml or Json Sending data to in server

**XMLHttpRequest**

1. Sends data from the client in the background
2. Receives the data from the server
3. Updates the webpage without reloading it.
4. **Making the Request in object**

Var https= new XMLHttpRequest();

1. **Onreadystatechange**

This method create the one function and allowes to change the dom

Content.

1. **readyState**

Holds the status of the XMLHttpRequest.

0: request not initialized   
1: server connection established  
2: request received   
3: processing request   
4: request finished and response is ready

1. **status**

Returns the status-number of a request

200: "OK"  
403: "Forbidden"  
404: "Not Found"

1. **this.readystate==4 && this.status==200**

Above condtion is true the data is fetch on UI

* + 1. **https.open(method,“url”,Asynchronous);**
    2. xhttp.send();

**send request to server to GET requests**

function loadDoc() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById("demo").innerHTML = this.responseText;

}

};

xhttp.open("GET", "ajax\_info.txt", true);

xhttp.send();

}

**This keyword in Js**

<https://www.tutorialsteacher.com/javascript/this-keyword-in-javascript>

This keyword is a global object.

Always refer to the globa object in function.

1. **Global object**

With in normal function **this keyword always refer global object.**

Nested functions **itself also refer global object.**

**Example:**

Var obj= 1;

function add1()

{

var obj= 11;

console.log(this.obj);///1

function add2()

{

var obj= 222;

console.log(this.obj);////1

}

add2();

}

add1();

1. **Objects method**

When you creating constructor **function using new keyword it refer**

**That particular object only**. Not global object.

var myVar = 100;

function WhoIsThis() {

this.myVar = 200;

}

Var x= new WhoIsThis();

**Refers the own object value not global object**

**How working Call, apply, bind method pf this**

Person.call=== it call the function and execute

Person.call(this,first)== it call the function and execute

This is refers to the own object

Of aruguments ex. first

var myVar = 100;

function WhoIsThis() {

alert(this.myVar);

}

var obj1 = { myVar : 200 , whoIsThis: WhoIsThis };

var obj2 = { myVar : 300 , whoIsThis: WhoIsThis };

WhoIsThis(); // 100

WhoIsThis.call(obj1); // 200

WhoIsThis.apply(obj2); // 300

obj1.whoIsThis.call(window); // 100

WhoIsThis.apply(obj2); // 300

**Function**

<https://www.freecodecamp.org/news/when-to-use-a-function-declarations-vs-a-function-expression-70f15152a0a0/>

<https://medium.com/tfogo/advantages-and-pitfalls-of-arrow-functions-a16f0835799e>

Function means when some events trigger it will execute some specified

Action.

Function expressions are best for object methods. Arrow functions are best for callbacks or methods like map, reduce, or forEach.

Var newFun = new Function(a,b);

newFun(5,10);

**Types:**

1. Function declaration **= name function**
   * + Function declaration suitable object method
     + Load before any code is executed  so that know is ontime of hoisting.
2. Function expression= **without name function**

* when the interpreter reaches that line of code execute. So cannot perform hoisting.
* It is mostly used for closures.

1. **Arrow functions**

* Using arrow function perform mostly iterating purpose.
* Arrow function also function expression.
* Arrow function not suitable object method. Also it don’t have **this** keyword
* Also arrow function not hoisted.
* Arrow function reduce code. No need **return** and **function** keyword.

1. **Function declaration**

**It will start only the function name.**

Function add()

{

//statements

}

1. **Function expression and self invoking function**

**Without name function**

1. Var a= function(){  
    //statements

}

1. (function(){

//statements

}());

|  |  |
| --- | --- |
| **Normal function declaration** function add()  {  return a;  }  add(); | **Arrow function**  **Arrow function is an function expression.** var arrow= (a)=>{a}; |

**Arrow function declarations:**

1. **Without parameter**

(()=>  
{  
console.log(“hi”);  
})();

1. **Withone parameter**

Var arrow= (a)=> {};

1. **Object literal**

(x)=> ({

})

1. Using arrow function we can iterate the array objects values.

var obj2= [{

name:"prabu",

age: 10

},

{

name:"prabu11",

age: 1110

}

];

obj2.forEach((ww)=>

{

console.log(ww.name);

}

)

**Promises:**

Using for Asynchronous operation in js.

Promises is an take two arguments.

1. Resolve
2. Reject

<https://www.geeksforgeeks.org/javascript-promises/>

var **promise** = new Promise(function(resolve, reject) {

  const x = "geeksforgeeks";

  const y = "geeksforgeeks"

  if(x === y) {

    resolve();

  } else {

    reject();

  }

});

**Promise**.then(function () {

        console.log('Success, You are a GEEK');

    }).

    catch(function () {

        console.log('Some error has occured');

    });

**Es6 Features:**

1. Class method = we can also extends the keywords the class using **Extends** Keyword
2. Also they introduce Import and export keywords.
3. Arrow functions
4. Block scopes
5. Let
6. Const
7. Promises
8. Array methods:
9. Find
10. FindIndex

**Es5 Features**

**Array methods:**

1. Some
2. Every
3. Filter
4. Map
5. ForEach
6. indexOf
7. LastIndexOf
8. JSON.parse
9. JSON.stringfy

**String methods:**

1. String.trim()

**Use strict mode.**