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**

**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 bubbling

Capturing = third parameter Add **true**

**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>

**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);

**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**

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:**

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

**Shallow copy method3:**

Spread Opertor

Var obj1= {…obj};

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));**

**Object Methods:**

1. **Object.assign()**

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 wheather object is sealed or not.