**Class based oop in javascript**

* Object" is a "physical item", which is a collection of properties (details) and methods (manipulations).
* "Class" is a "model" of objects, which defines the list of properties and methods of the objects.

**Steps for working with Classes and Objects**

**Create a class**

class classname

{

}

**Create an object**

new classname();

**Example on class and object**

**<script>**

**Class Student{**

**}**

**Var t=new Student();**

**Console.log(t);**

**</script>**

**Constructor**

-----------------------------------

* Constructor is a special method in the class, that executes every time when we created an object for the class.
* Constructor's name should be "constructor".
* Constructor can receive arguments; but can't return any value.
* It is not possible to call the constructor without creating an object.
* Constructors are mainly used to create an initialize properties in the class.
* Parameterized Constructor is a constructor that receives one or more arguments (parameters) and initializes the same to the respective properties.

**Syntax of Constructor**

class classname

{

constructor(arguments)

{

}

}

**Example on Constructors**

**<script>**

**Class Student{**

**Constructor(){**

**this.sno=101**

**this.sname=”Scott”;**

**}**

**}**

**Var obj=new Student();**

**Console.log(obj.sno+” ”obj.sname);**

**</script>**

**Param Constructors and methods**

**<script>**

**Class Student{**

**Constructor(sno,sname,marks){**

**this.sno=sno;**

**this.sname=sname;**

**this.marks=0;**

**}**

**result(){**

**if(this.marks>=35){**

**return “pass”;**

**}else{**

**return “fail”;**

**}**

**}**

**}**

**Var obj=new Student(101,”Ajay”);**

**Console.log(obj.sno);**

**Console.log(obj.sname);**

**Console.log(Obj.result());**

**</script>**

**Inheritance**

* Inheritance is a concept of extending the parent class, by creating the child class.



* When you create an object of child class, all the members of parent class will be automatically added to the child class's object.

**Syntax of Inheritance**

class parentclassname

{

constructor(arguments){

}

}

class childclassname extends parentclassname

{

constructor(arguments)

{

super(arguments);

}

}

**Example on Inheritance**

**<script>**

**class Person{**

**constructor(no,name){**

**this.no=no;**

**this.name=name;**

**}**

**}**

**Class Student extends Person{**

**Constructor(no,name,sid,marks){**

**Super(no,name);**

**This.sid=sid;**

**This.marks=marks;**

**}**

**}**

**</script>**

**Method overriding**

* Method Overriding is a concept of extending the "parent class method", by creating similar method in the "child class", with same signature (name and arguments).

**Syntax of Method Overriding**

class parentclassname

{

method(arguments)

{

}

}

class childclassname extends parentclassname

{

method(arguments)

{

super.method(arguments);

}

}

**Destructuring**

Destructuring is used to retrieve each value of an array into respective variables.

**Steps for working with Destructuring**

var [ variable1, variable2, …] = arrayname;

**Example 1 on Destructuring**

<script>

Var cities=[“Hyd”,”chennia”,”Bangalore”,”pune”,”mombai”];

Var [city1,city2,,city4]=cities;

Console.log(city1);

Console.log(city2);

Console.log(city4);

</script>

**Let**

* The "let" keyword can be used as alternative for "var" keyword, to create variables.
* The "var" keyword always creates "local variables" or "block level variables"; but "let" keyword creates "block level variables".

**Steps for working with Let**

let variablename = value; **UI Technologies**

**Example on Let**

**<script>**

**for(let i=0;i<=10;i++){**

**}**

**Console.log(i);**

**</script>**

**Var and Let**

<!DOCTYPE html>

<html>

<head>

<script>

const g=1232;

var i=3;

g=22;

for(var i=0;i<10;i++){

console.log(i);

}

console.log("var "+i);//no block scope for var-keyword

/\*\*\*\*\*\*\*\*\*\*\*\*\*Let keyword\*\*\*\*\*\*\*\*\*\*\*\*\*/

let j=2;

for(let j=0;j<10;j++){

console.log(j);

}

console.log("let "+j);//block scope works here with let-keyword

</script>

</head>

<body>

</body>

</html>

**Const**

* The "const" keyword is used to create constants.
* We can't change the value of the constant.
* If you try to change the value of constant, it shows error.

**Steps for working with Const**

**<script>**

**const v=10;**

**console.log(v);**

**</script>**

**undefined vs null**

* "undefined" represents "empty value", which is by default to assigned to every uninitialized variables. The developer is not supposed to assign "undefined" manually.

• "null" represents "empty value", which can be assigned by the developer.

• The datatype of "undefined" is "undefined".

• The datatype of "null" is "object".

== and ===

* 1. == operator checks only value; === operator checks value and data type also.
  2. == operator internally first converts the right side value in the data type of left side and checks the value. === operator will not perform any automatic conversion and directly checks the value.
  3. • Use == operator to check only value. Use === operator check value and data type

**Syntax of ==** value1 == value2

**Syntax of ===** value1 === value2

**document.getElementsByTagName()**

* This method retrieves the array of elements that have specified tag name.

**Syntax:** document.getElementsByTagName("tag name")

Ex:

<!DOCTYPE html>

<html>

<body>

First Name: <input name="fname" type="text" value="Michael"><br>

<select name="cnt">

<option>India</option>

<option>India</option>

<option>India</option>

</select>

<input type="radio" name="gender"/>Male

<button onclick="myFun()">click it</button>

<script>

function myFun() {

var x = document.getElementsByName("gender")[0].tagName;

alert(x);

}

</script>

</body>

</html>

……………………………………………………

**Reading Elements from Array**

* The most common task of the developer is reading data from an array and displaying the same.
* This can be done in several ways:

1. For Loop

2. ForEach Loop

3. ForOf Loop

4. ForIn Loop

**Steps for working with For Loop**

for (i = 0; i < array.length; i++)

{

array[i]

}

EX:

var c={12,13,15,16};

for(i=0;i<c.length;i++){

console.log(c[i]);

}

* For loop is index based.
* Forward / backward iterations are possible
* We can start the loop from the middle also.

**Steps for working with ForEach Loop**

array.forEach( function(variable)

{

variable

} );

* For loop is value based. The variable contains "value" directly; but not index.
* Forward iterations are only possible. Backward iterations are not possible.
* We can't start the loop from the middle.

**Steps for working with ForOf Loop**

for (variable of array)

{

variable

}

• For loop is value based. The variable contains "value" directly; but not index.

* Forward iterations are only possible. Backward iterations are not possible.
* We can't start the loop from the middle.

**Steps for working with ForIn Loop**

for (variable in array)

{

array[variable]

}

* For loop is index based. The variable contains "index".
* Forward iterations are only possible. Backward iterations are not possible.
* We can't start the loop from the middle.

**Object Oriented Programming (OOP) languages**

**Types of Object Oriented Programming (OOP) languages**

We have two types of OOP languages:

1. Class-based Object Oriented Programming Language

2. Prototype-based Object Oriented Programming Language

**Creating Objects**

We can create object in 2 ways:

1. Object Literals

2. Constructor Function

**Object Literals**

Object literals are represented as curly braces { }, which can include properties and methods.

The property and value are separated with : symbol.

**Syntax:** { “property”: value, “method”: function( ) { … } }

Ex: Object litral

<script>

var std={studentid:1,studentname:”scott”,marks:80};

console.log(std);

console.log(std.studentid);

console.log(std.studentid);

console.log(std.studentname);

console.log(std.marks);

</script>

Ex:

<script>

Var std={studentid:1,studentname:”scott”,marks:30,”result”:function(){

If(this.marks>=35){

return “pass”;

}else{

Return “fail”;}

}

};

console.log(std);

console.log(std.studentid);

console.log(std.studentname);

console.log(std.marks);

console.log(std.result());

</script>

**Constructor function**

* Constructor function is a function that receives an empty (new) object, initializes properties and methods to the object.
* • The "this" keyword inside the constructor function represents the current working object. For example, if it is called for the first time, the "this" keyword represents the first object; if it is called second time, the "this" keyword represents the second object.
* • The constructor function can receive one or more parameters and initializes the same values into the respective properties.
* • The "reference variable" stores the reference (address) of the object and used to access its members (properties and methods), outside the object literal / constructor function.
* **Syntax of Constructor Function**
* function functionname(arguments)
* {
* this.property = value;
* this.method = function( ) { … };
* }
* var variablename = new functionname( );
* **Example on Constructor Function**

function Student(sno,sname,marks){

this.sno=sno;

this.sname=sname;

this.marks=function(){

if(this.marks>=30){

return “pass”;

}else{

return “fail”;

};

}

var std=new Student(101,”scott”,230);

console.log(std);

console.log(std.sno);

console.log(std.sname);

console.log(std.marks);

}

**Object.Keys**

* • The “Object.keys” method is used to retrieve the list of properties of an object as an array.
* • Sometimes, you will get data from server / browser storage. Then you don't know what properties / methods are present inside the object. Then you have to use Object.keys() are used to properties / methods of the object and also read its values programmatically.

**Syntax:** Object.keys(reference variable);

**Example:** Object.keys(stu);

* • This is useful if you don’t know what properties are exist in the object.

Example:

const object1 = {

a: 'somestring',

b: 42,

c: false

};

console.log(Object.keys(object1));

Arrow Function

Arrow functions were introduced in ES6.

Arrow functions allow us to write shorter function syntax:

<!DOCTYPE html>

<html>

<body>

<h2>With "use strict":</h2>

<h3>Using a variable without declaring it, is not allowed.</h3>

<p>Activate debugging in your browser (F12) to see the error report.</p>

<script>

"use strict";

x = 3.14; // This will cause an error (x is not defined).

</script>

</body>

</html>

# JavaScript Function Call

## **Method Reuse**

With the call() method, you can write a method that can be used on different objects.

## **All Functions are Methods**

In JavaScript all functions are object methods.

If a function is not a method of a JavaScript object, it is a function of the global object (see previous chapter).

The example below creates an object with 3 properties, firstName, lastName, fullName.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Functions</h2>

<p>This example creates an object with 3 properties (firstName, lastName, fullName).</p>

<p>The fullName property is a method:</p>

<p id="demo"></p>

<script>

var myObject = {

firstName:"John",

lastName: "Doe",

fullName: function() {

return this.firstName + " " + this.lastName;

}

}

x = myObject.fullName();

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

