Object Oriented Programming in JavaScript

Object Oriented Programming (OOP) is a programming paradigm (programming structure), which is based on the concept of “object”.

* Object represents a physical component/real-time entity.
  + We can see
  + We can touch
  + We can use
* Object is a collection of two types of members:

1. variables or fields (**properties**)
2. functions **(methods**)

* **Properties or fields:** details about the object. Properties are the variables stored inside the

Object. Properties are used to store data about specific person, product or thing.

* **Methods:** to perform manipulations on the properties. Methods are the functions stored inside the object. **Methods** **read values from properties**, **write values into properties,** to **perform** **logical operations.**

**Example:**

Car is an object: Person is an object:

-properties -properties

* Car model: I20 > name: siva
* Car colour : white > age: 50
* Car no: 5579 > gen: male

-methods -methods

* Start() > sleep()
* Change gear() > eat()
* Stop() > walk()
* In the above example the “car” object has three properties called “car model, car colour, car no”, which have respective values.
* We have two types of OOP languages:

1. class-based Object-Oriented Programming

ex: java, .net, python, cpp etc…

1. prototype-based Object-Oriented Programming

ex: javascript, typescript, vbscript, perl,

Student st = new Student();

St = new Student(); 🡺 prop, method

Object 🡺 object 🡸 adding to

**Creating objects:**

we can create objects in 2 ways:-

1. with object literals
2. by using constructor function

**Object literals**

* Object literals are represented as curly braces **{ }**, which can include properties and methods.
* The property and values are separated with : symbol
* The method-name and body are separated with : symbol

Syntax:

let refname = {“property” **:** value, property **:** value, ...,

“method-name”: function() { steps },

"method-name": function(args) { steps } };

how to access?

object.property

object.property=value

object.method-name()

**Example 1 on object literals (properties)**

<Html>

<head>

<title> object literals </title>

</head>

<h1> object literals</h1>

<script>

var stu1= {"id" : 1, name : "ram", marks: 80};

var stu2= {id : 2, name : "sam", marks: 90};

document.write("Id :"+stu1.id+"<br>");

document.write("Name :"+ stu1.name+"<br>");

document.write("Marks :"+ stu1.marks+"<br>");

document.write(stu1+"<br><br>");

document.write("Id :"+ stu2.id+"<br>");

document.write("Name :"+ stu2.name+"<br>");

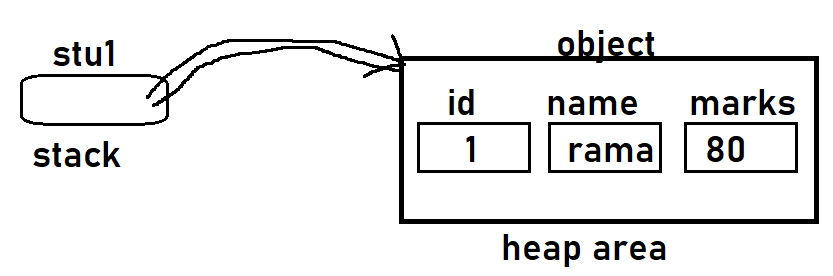
document.write("Marks :"+ stu2.marks+"<br>");

document.write(stu2);

</script>

</body>

</html>



**Example 2 oncreating object with literals**

<Html>

<head>

<title>object literals</title>

</head>

<body>

<h1> object with properties and methods </h1>

<script>

var stu1= { "id" : 1, name : "ram", marks: 30, "getResult": function() {

if(stu1.marks>=40)

return "Pass";

else

return "Fail";

},

};

document.write("Id :"+ stu1.id +"<br>");

document.write("Name :"+ stu1.name +"<br>");

document.write("Marks :"+ stu1.marks +"<br>");

document.write("Result is :"+ stu1.getResult() +"<br>");

document.write(stu1+"<br><br>");

</script>

</body>

</html>

**Constructor function**

Constructor 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” key word represents the first object; if it is called second time, it represents the second object.

Constructor functions technically are regular functions. There are two conventions though:

1. They are named with capital letter first.
2. They should be executed only with "new" operator.

**Syn:**

**function Constname() 🡸 constructor developing**

**{**

**this.property1=value; 🡸 initializing code**

**this.property2=value;**

**...**

**this.method-name = function(){**

**code**

**};**

**this.method-name = function(){**

**code**

**};**

**...**

**}**

**Lit: no iden : , no this**

**Const: iden = ; this**

**Var refname = new Const-funname(); 🡸 constructor calling**

**Var refname = new Const-funname(args);**

**Note:** constructor functions don’t return any value, hence no return statement.

**<!-- example on creating object with constructor -->**

<Html>

<head>

<title>constructor</title>

</head>

<body>

<h1> object with constructor </h1>

<script>

function Book(name, year)

{

this.name = name;

this.year = '(' + year + ')';

}

var firstBook = new Book("Html", 2014);

var secondBook = new Book("JavaScript", 2013);

var thirdBook = new Book("CSS", 2010);

document.write(firstBook.name, firstBook.year + "<br>");

document.write(secondBook.name, secondBook.year + "<br>");

document.write(thirdBook.name, thirdBook.year + "<br>");

</script>

</body>

</html>

**<!-- example on creating object with constructor -->**

<Html>

<head>

<title>Constructor</title>

</head>

<body>

<h1> Constructor with properties and methods </h1>

<script>

function Student(id,name,total)

{

//initializing

this.id=id;

this.name=name;

this.total=total;

this.getResult = function() {

if(this.total>=40)

return "Pass";

else

return "Fail";

}; //end of method

} //end of const

var stu = new Student(11, "Ram", 88);

document.write("Id :"+ stu.id +"<br>");

document.write("Name :"+ stu.name +"<br>");

document.write("Marks :"+ stu.total +"<br>");

document.write("Result is :"+ stu.getResult() +"<br>");

</script>

</body>

</html>

**Using the new keyword is essential**

It's important to remember to use the new keyword before all constructors. If you accidentally forget new, you will be modifying the global object instead of the newly created object. Consider the following example:

**<!-- example on this & instanceof keywrod -->**

<html>

<head>

<title>Document</title>

</head>

<body>

<script>

function Book(name, year)

{

console.log(this);

this.name = name;

this.year = year;

}

var myBook = Book("js book", 2014);

console.log(myBook instanceof Book);

console.log(window.name, window.year);

var myBook = new Book("js book", 2014);

console.log(myBook instanceof Book);

console.log(myBook.name, myBook.year);

</script>

</body>

</html>

**<!-- Object Array Literals -->**

<Html>

<body>

<h1> Creating Object Array with Literals </h1>

<script>

//object array

var emps =[ { id:11, name:"ram", sal:35000 },

{ id:22, name:"sam", sal:45000 },

{ id:33, name:"rahim", sal:25000 }

];

//retrieving data from array

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

document.write(emps[i].id, emps[i].name, emps[i].sal+"<br>");

}

document.write(emps);

</script>

</body>

</html>

**<!-- exmaple on object arrays -->**

<html>

<head>

<script>

function totalValue(prods) //user define function

{

let inventory\_value = 0;

for(let i=0; i<prods.length; i+=1) {

amt= prods[i].inventory \* prods[i].unit\_price;

inventory\_value +=amt;

document.write(prods[i].name, amt, "<br>");

}

return inventory\_value;

}

</script>

</head>

<body>

<script>

let products = [ { name: "chair", inventory: 5, unit\_price: 45},

{ name: "table", inventory: 10, unit\_price: 120},

{ name: "sofa", inventory: 2, unit\_price: 500}

];

//array passing as an param to fun

document.write("Total Bill Amt :"+ totalValue(products) );

</script>

</bodY>

</html>

**<!-- exmaple on object arrays with constructor -->**

<html>

<body>

<h1> Creating Object Array with Constructor Function </h1>

<script>

function Movie(name,hero,dir) //constructor

{

this.name=name;

this.hero=hero;

this.dir=dir;

}

//object array

var movies=[ new Movie("Bharath","Mahesh","Siva"),

new Movie("ASVR","Ntr","Trivikram")

];

//retrieving data from array

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

document.write(movies[i].name, movies[i].hero, movies[i].dir+"<br>");

}

document.write(movies);

</script>

</body>

</html>

**prototype**

the "prototype" generally represents model of the object (structure), which contains list of properties and methods of the object.

**"prototype" is a predefine attribute.**

All JavaScript objects inherit properties and methods from a prototype:

* Student objects inherit from Student.prototype
* Date objects inherit from Date.prototype
* Array objects inherit from Array.prototype

The **Object.prototype** is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from **Object.prototype**.

**Adding Properties and Methods to Objects:**

Sometimes you want to add new **properties or methods** to an existing object literal of a given type.

Sometimes you want to add new **properties or methods** to an object constructor.

**Syn:**

**Constructor Syn:**

**Constructor.prototype.new-property = value;**

**Constructor.prototype.new-method = function() { code };**

**Literal Syn:**

**Object**.**prototype.new-property = value;**

**Object.prototype.new-method = function() { code };**

**<!-- exmaple on prototype -->**

<html>

<body>

<script>

function Product(name,qty,unitPrice) //constructor function (class)

{

this.name=name;

this.qty=qty;

this.unitPrice=unitPrice;

}

//adding new property to an existing object

Product.prototype.discount=10;

//adding new method to an existing object

Product.prototype.getAmount=function(){

return this.qty\*this.unitPrice;

};

//creating object

let p = new Product("Soap",2,42.50);

document.write("Name :"+ p.name +"<br>");

document.write("Qty :"+ p.qty +"<br>");

document.write("UnitPrice :"+ p.unitPrice +"<br>");

document.write("TotalAmt :"+ p.getAmount() +"<br>");

document.write("Discount :"+ p.discount +"<br>");

document.write("BillingAmt :"+ (p.getAmount()-p.discount) +"<br>");

</script>

</bodY>

</html>

**Inheritance**

> the process of creating an object based on another object (prototype) is called as "inheritance".

> hence all the properties and methods of the 1st object (parent) is inherited into the 2nd object (child).

> by calling 1st object's constructor from 2nd object's constructor function.

**Syn:**

**function ConstructorP(parameters) //parent**

**{**

**properties**

**methods**

**}**

**function ConstructorC(parameters) //child**

**{**

**ConstructorP(); 🡸 inheritance**

**OR**

**ConstructorP.call(this, parameters); 🡸 inheritance**

**properties**

**methods**

**//here we access CP properties & methods directly**

**}**

**call()**

call() is a predefine function, it's used to call parent constructor function from child constructor function.