Object Oriented Programming in JavaScript

We have two types of OOP languages:

1. **class-based** Object-Oriented Programming

ex: java, .net, python, cpp etc…

1. **prototype-based (object based)** Object-Oriented Programming

ex: javascript, typescript, perl, php, …

* Object represents a physical component.
* Object is a real-time entity.
  + We can see
  + We can touch
  + We can use

Ex: Shopping bagCakeTelephoneAlarm clockSoldierNeedleetc…

* Object is an instance of a class, nothing but memory block (one copy of class)

“**Object**” is a predefine class; every class/object should be derived from “**Object**” class prototype.

* Object is a collection of members:

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

* **Properties:** details about the object. Properties are the variables which are stored inside the Object. Properties are used to store data about specific person,product or thing.

Ex: array.length=5

* **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.**

Ex: array.sort() array.push() array.indexOf()

**Note:** objects are used to data maintenance

**Array Object**

Sequence random

Index base properties (key)

**[ ] { }**

**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() > talk()
* In the above example the “car” object has three properties called “car model, car colour, car no”, which have respective values.

“**Object**” is a predefine class; every class/object should be derived from “**Object**” class prototype.

**Creating objects:**

We can create objects in 2 ways: -

1. with object literals
2. by using new & 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:

const refname = {

“property”**:**value, property**:**value, ...,

“method-name”**:** function() {body},

method-name() {body}

}

Note:

Methods we can write in any of 3 forms, i.e like normal function or expression or arrow

how to access?

refname.property

refname.property=value

refname.method-name()

**Note:**

**Object** is one of predefined class.

Every js class and every js object should be derived from an Object class **prototype**.

**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>");

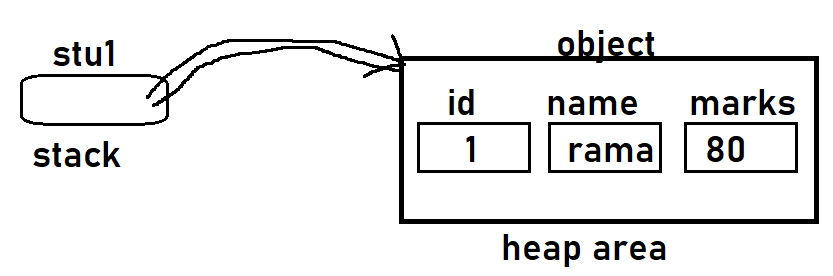
document.write(stu2);

</script>

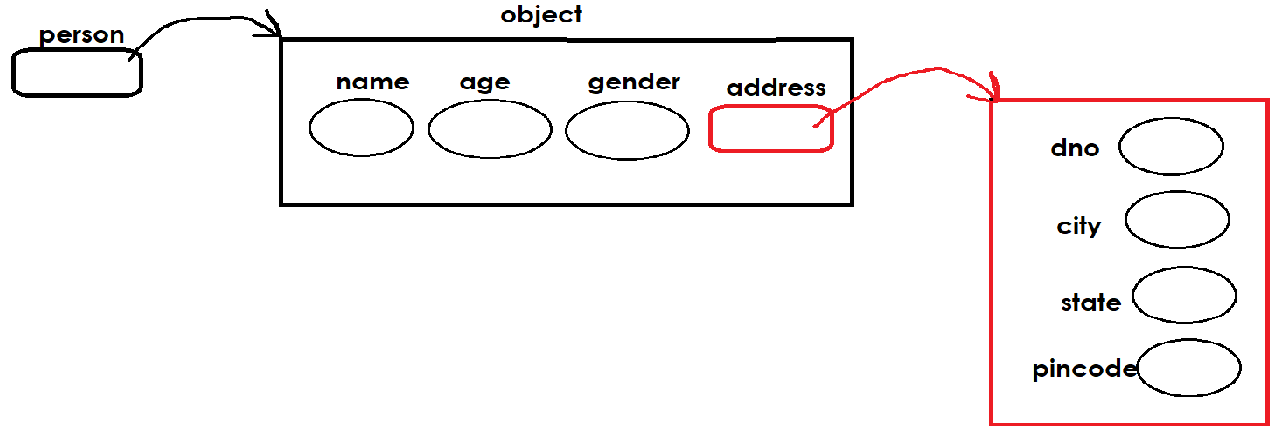
</body>

</html>







****

**Example 2 on creating 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>

**This keyword**

The “this” keyword represents/substitutes 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.

"this" is a keyword

"this" is a predefined reference variable (Same as to stud )

"this" maintains reference of current working object.

"Stud" what it maintains "this" also maintains the same thing

**"this" used within the object, but "stud" used outside the object**

If you want access members within the object use "this".

If you want access members outside the object use "stud"

**Note:** If we want access properties inside method or constructor function, we should use “this” keyword.

**Constructor function**

Constructor is a function that receives an empty (created by new keyword) object, initializes properties and methods to the object (object init).

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

1. They are named with capital letter first.
2. They should be executed only with "new" keyword, while object creation.
3. Constructor functions don’t return any value, hence no return statement.

**Syn**

**function Constname(params)**

**{**

**this.prop=value;**

**this.prop=value;**

**…**

**methods…**

**}**

**Object Syn:**

**const refname = new Constname(); 🡸 constructor calling**

**const refname = new Const-name(args);**

**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:

**<!-- 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}

];

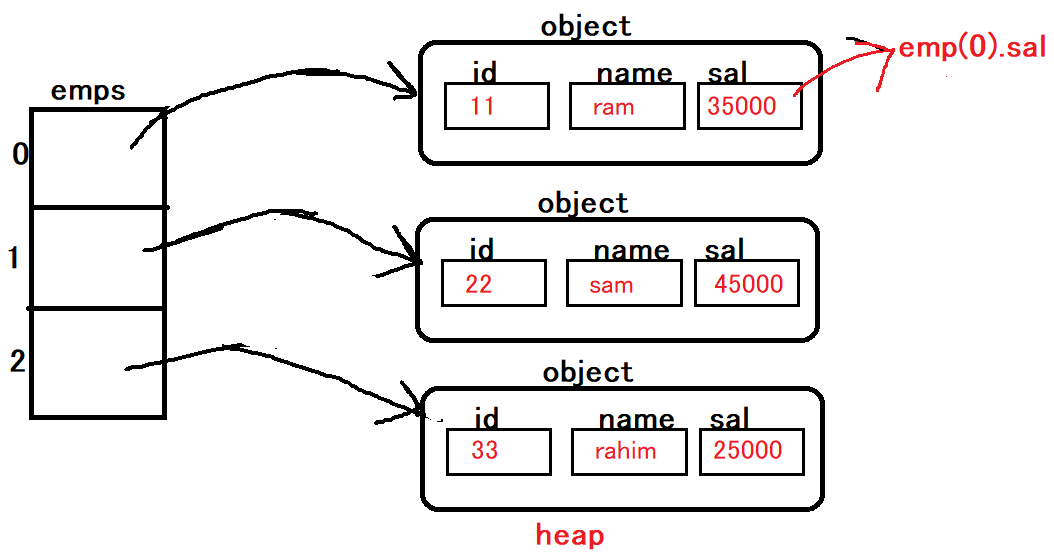
//array passing as an param to fun

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

</script>

</bodY>

</html>





**prototype**

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

**"prototype" is a predefine property.**

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 };**

refname.new-prop=value

refname.new-method=body

**Functional Inheritance**

> The process of creating a new object based on another object (exists) is called "Functional inheritance".

OR

> The process of creating a new constructor based on another constructor (exists) is called "Functional inheritance".

> used to build the relationship between multiple constructors

> used to extending functionalities of existing constructor without modifying source code.

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

> sharing

> by calling 1st constructor from 2nd constructor function using call().

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

Syn:

function Constructor(params) //Parent

{

Prop init

Methods  
 }

function Constructor(params) //Child

{

**Constructor.call() or Constructor.call(this, params)**

Prop init

Methods

}

refvar = new Constructor();

1 constructor 🡸 parent/base

Prop

Methods

New constr 🡸 child/sub

Prop

methods

**Inheritance**

> the process of creating a new class based on another class (exists) is called "inheritance".

> used to build the relationship between multiple classes

> used to extending functionalities of existing classes without modifying source code.

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

> sharing

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

**Syn:**

**class ClassName1 //parent**

**{**

**constructor(parameters)**

**{**

**Properties init**

**}**

**methods**

**}**

**class ClassName2 extends ClassName1**

**{**

**constructor(parameters)**

**{**

**super();**

**properties init**

**}**

**methods**

**//here we access parent class properties & methods directly**

**}**

**super()**

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

**Javascript** classes are supports 3 types of inheritance, those are

1. Single level
2. Multilevel
3. Hierarchal