Objects in JavaScript

- In JavaScript, an object is a collection of properties, defined as a key-value pair. Each property has a key and a value. The property key can be a string and the property value can be any valid value.
- Properties can hold values of primitive data types and methods are functions.
- Functions inside of objects are known as Methods.
- We can store functions, arrays & even objects as objects properties.
- When we have to access the object properties inside that of object method we have to use 'this' keyword.
- In order to access or get the value of a "property", we can use the dot (.), or square bracket ([]) notation. Here dot(.) operator is powerfull than that of bracket([]) notation.

Ways to creating an object

In JavaScript, an object can be created in two ways :-

- Object literal
- Object constructor

Object Literals

- The object literal is a simple way of creating an object using { } brackets.
- We can include key-value pair in { }, where key would be property or method name and value will be value of property of any data type or a function.

- Use comma (,) to separate multiple key-value pairs.

Syntax

```
var objName = {
   key1: value1,
   key2: value2,
   ...
   keyN: valueN
};
```

Example

```
var abc = {
   name : "Mohit",
   age:34
}
console.log(abc.name)
```

Example

```
var abc = {
   name : "Mohit",
   age:34
}
console.log(abc['name'])
```

Example

We can use square braces to place two words as a key in that object.

```
var abc = {
   "Full Name" : "Mohit Singh",
   age:25}
console.log(abc['Full Name']);
```

Example -- Object as Reference

```
function person(obj){
  obj.name = "Kelvin"
}
var personInfo = {
  name : "Mohit"
}
person(personInfo);
console.log(personInfo.name);
```

Example -- With Reserved Words as Object properties

```
var person = {
    switch:"Hello Mohit",
    if:"Kelvin"
}
console.log(person.if);
```

Example -- Nested Arrays

```
var person = {
  name : "Mohit",
  age  : 34,
  nestObj : [10,20,"Hello",null]
}
console.log(person.nestObj[0])
```

Example -- Nested Objects

```
var person = {
  name : "Mohit",
  age  : 34,
  nestObj : {
    id : 1,
    gender : "Male"
  }
}
console.log(person.nestObj.gender)
```

Example -- Modifies Object Properties

```
var person = {
   fname:"Mohit",
   lname:"Singh"
}
console.log(person.fname);
person.fname = "Kelvin"
console.log(person.fname);
```

Example -- Deleting Object Properties

```
var person = {
   fname:"Mohit",
   lname:"Singh"
}
console.log(person.fname);
delete person.fname;
console.log(person.fname);
```

Object Constructor

- The second way to create an object is with Object Constructor using 'new' keyword.
- We can attach properties and methods using dot notation.
- We can also create properties using [ ] brackets and specifying property name as string.

Syntax

```
var objName = new Object();
```

Example

```
var person = new Object(); // Object Constructor
person.fname = "Mohit";
person.lname = "Singh";
console.log(person);
```

Example

```
var person = new Object();
person['fname'] = "Mohit";
person['lname'] = "Singh";
console.log(person)
```

Example

```
var person = Object.create({
   fname:"mohit",
   lname:"singh"
});
console.log("Hello "+person.fname+" "+person.lname);
```

creating object using create function() of Object

Objects Methods

- When we define a function as the object property then this type of function is known as Object Method.
- When we have to access the object properties inside of object method then we will use 'this' keyword.
- In simple words, Methods are just function define inside object as its properties.

Example

```
var abc = {
   "Full Name" : "Mohit Singh",
   age:34,
   func:function(){
      console.log("Hello Method")
   }
}
abc.func();
```

Example

```
var abc = {
   "Full Name" : "Mohit Singh",
   age:34,
   func:function(){
      console.log("Hello Method")
   }
}
abc['func']();
```

Example

```
var person = {};
person['fname'] = "Mohit";
person['lname'] = "Singh";
person['fullName'] = function(){
   return this.fname +" "+this.lname;
}
console.log(person['fullName']());
```

Example

```
var person = {}; // Object Literal
person.fname = "Mohit";
person.lname = "Singh";
person.fullName = function(){
   return this.fname +" "+this.lname;
}
console.log(person.fullName());
```

Example

```
var person = {
   fname:"Mohit",
   lname:"Singh"
}
person.fullName = function(){
   console.log(this.fname +" "+this.lname)
}
person.fullName();
```

Example

```
var person = {
   fname:"Mohit",
   lname:"Singh",
   age:34,
   info:function(){
      console.log(this.fname)
   }
}
console.log(person.fname);
console.log(person.lname);
console.log(person.age);
person.info();
```

Example

```
var person = {
   fname:"Mohit",
   lname:"Singh",
   age:34,
   info:function(){
      console.log(this.fname)
   }
}
console.log(person['fname']);
console.log(person['lname']);
console.log(person['age']);
person['info']();
```

Example

```
var person = {};
person.fname = "Mohit";
person.lname = "Singh";
person.info = function(){
   console.log("Hey I am Method Of Person Object");
}
console.log(person);
```

Example

```
var person = {};
person['fname'] = "Mohit";
person['lname'] = "Singh";
person['info'] = function(){
   console.log("Hey I am Method Of Person Object");
}
console.log(person);
```

Checking Object Property Existence

If we wants to check that an object has a particular property or not, then we should use 'hasOwnProperty()' method before accessing properties.

Example

```
var person = {};
console.log(person.noProp === undefined)
```

Example

```
var person = {};
console.log('fname' in person);
```

Example

```
var person = new Object();
person.fullName = "Mohit Singh";
console.log(person.hasOwnProperty('fullName'));
console.log(person.hasOwnProperty('fname'));
console.log(person.hasOwnProperty('lname'));
```

Example

```
var fruits = {
  fruit1 : "Mango",
  fruit2 : "Grapes",
  fruit3 : "Orange"
}
for (var property in fruits) {
        if (fruits.hasOwnProperty("fruit1")) {
          console.log("Property Exists");
          break;
  }
  else  {
    console.log("Property Not Found");
    break;
  }
}
```

Private Properties & Methods in JavaScript Objects

- Normally our object properties are public, hence we can access them from anywhere else.
- In order to make object properties and methods private then we have to declare them by using var, let keywords.

Example

```
var person = function(fname,lname,age){
    this.fname = fname;
    this.lname = lname;
    var age = age;
    this.accessPrivateProp = function(){
        return age;
    }
}
var person1 = new person("Mohit","Singh",34)
console.log(person1.accessPrivateProp());
```

Looping Objects

We can use "for .. in" loop in order to access all the properties & values of objects.

Example -- Looping Object

```
var objName = {
  name : "Mohit",
  age :  34
}
for(var i in objName){
  console.log(i)
}
```

Example -- Accessing Keys & Values

```
var personInfo = {
  name : "Mohit",
  age : 34,
}
for(i in personInfo){
  console.log(i+" : "+personInfo[i])
}
```

Example -- Looping Object

```
var person = new Object();
person.firstName = "Mohit";
person["lastName"] = "Singh";
person.age = 34;
person.getFullName = function () {
      return this.firstName + ' ' + this.lastName;
};
for(var key in person){
      console.log(key);
};
```

Example -- Looping 'Object.entries()'

```
var fruits = {
  fruit1 : "Mango",
  fruit2 : "Grapes",
  fruit3 : "Orange"
}
var f = Object.entries(fruits)
for (x of f) {
  console.log(x);
}
```

Example -- Cloning Object

```
let user = {
  name: "Mohit",
  age: 25
};
let clone = {};
for (let key in user) {
  clone[key] = user[key];
}
clone.name = "Kelvin";
alert( user.name );
```