***Blog : Objects and its internal representation in Javascript.***

Objects in JavaScript, just as in many other programming languages, can be compared to objects in real life. The concept of objects in JavaScript can be understood with real life, tangible objects.

In JavaScript, an object is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.

Objects and properties

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation:

objectName.propertyName

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let’s create an object named myCar and give it properties named make, model, and year as follows:

var myCar = new Object();  
myCar.make = 'Ford';  
myCar.model = 'Mustang';  
myCar.year = 1969;

Unassigned properties of an object are [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) (and not [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null)).

myCar.color; // undefined

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see [property accessors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors)). Objects are sometimes called *associative arrays*, since each property is associated with a string value that can be used to access it. So, for example, you could access the properties of the myCar object as follows:

myCar['make'] = 'Ford';  
myCar['model'] = 'Mustang';  
myCar['year'] = 1969;

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime). Examples are as follows:

// four variables are created and assigned in a single go,   
// separated by commas  
var myObj = new Object(),  
 str = 'myString',  
 rand = Math.random(),  
 obj = new Object();myObj.type = 'Dot syntax';  
myObj['date created'] = 'String with space';  
myObj[str] = 'String value';  
myObj[rand] = 'Random Number';  
myObj[obj] = 'Object';  
myObj[''] = 'Even an empty string';console.log(myObj);

Please note that all keys in the square bracket notation are converted to string unless they’re Symbols, since JavaScript object property names (keys) can only be strings or Symbols (at some point, private names will also be added as the [class fields proposal](https://github.com/tc39/proposal-class-fields) progresses, but you won’t use them with [] form). For example, in the above code, when the key obj is added to the myObj, JavaScript will call the [obj.toString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString" \t "_blank) method, and use this result string as the new key.

You can also access properties by using a string value that is stored in a variable:

var propertyName = 'make';  
myCar[propertyName] = 'Ford';propertyName = 'model';  
myCar[propertyName] = 'Mustang';

You can use the bracket notation with [for...in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) to iterate over all the enumerable properties of an object. To illustrate how this works, the following function displays the properties of the object when you pass the object and the object's name as arguments to the function:

function showProps(obj, objName) {  
 var result = ``;  
 for (var i in obj) {  
 // obj.hasOwnProperty() is used to filter out properties from the object's prototype chain  
 if (obj.hasOwnProperty(i)) {  
 result += `${objName}.${i} = ${obj[i]}\n`;  
 }  
 }  
 return result;  
}

So, the function call showProps(myCar, "myCar") would return the following:

myCar.make = Ford  
myCar.model = Mustang  
myCar.year = 1969

Enumerate the properties of an object

Starting with [ECMAScript 5](https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/ECMAScript_5_support_in_Mozilla), there are three native ways to list/traverse object properties:

* [for...in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) loops  
  This method traverses all enumerable properties of an object and its prototype chain
* [Object.keys(o)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys)  
  This method returns an array with all the own (not in the prototype chain) enumerable properties' names ("keys") of an object o.
* [Object.getOwnPropertyNames(o)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyNames)  
  This method returns an array containing all own properties' names (enumerable or not) of an object o.

Before ECMAScript 5, there was no native way to list all properties of an object. However, this can be achieved with the following function:

function listAllProperties(o) {  
 var objectToInspect;   
 var result = [];  
   
 for(objectToInspect = o; objectToInspect !== null;   
 objectToInspect = Object.getPrototypeOf(objectToInspect)) {   
 result = result.concat(  
 Object.getOwnPropertyNames(objectToInspect)  
 );   
 }  
   
 return result;   
}

This can be useful to reveal “hidden” properties (properties in the prototype chain which are not accessible through the object, because another property has the same name earlier in the prototype chain). Listing accessible properties only can easily be done by removing duplicates in the array.

Creating new objects

JavaScript has a number of predefined objects. In addition, you can create your own objects. You can create an object using an [object initializer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer). Alternatively, you can first create a constructor function and then instantiate an object invoking that function in conjunction with the new operator.

Using object initializers

In addition to creating objects using a constructor function, you can create objects using an [object initializer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer). Using object initializers is sometimes referred to as creating objects with literal notation. “Object initializer” is consistent with the terminology used by C++.

The syntax for an object using an object initializer is:

var obj = { property\_1: value\_1, // property\_# may be an identifier...  
 2: value\_2, // or a number...  
 // ...,  
 'property n': value\_n }; // or a string

where obj is the name of the new object, each property\_i is an identifier (either a name, a number, or a string literal), and each value\_i is an expression whose value is assigned to the property\_i. The obj and assignment is optional; if you do not need to refer to this object elsewhere, you do not need to assign it to a variable. (Note that you may need to wrap the object literal in parentheses if the object appears where a statement is expected, so as not to have the literal be confused with a block statement.)

Object initializers are expressions, and each object initializer results in a new object being created whenever the statement in which it appears is executed. Identical object initializers create distinct objects that will not compare to each other as equal. Objects are created as if a call to new Object() were made; that is, objects made from object literal expressions are instances of Object.

The following statement creates an object and assigns it to the variable x if and only if the expression cond is true:

if (cond) var x = {greeting: 'hi there'};

The following example creates myHonda with three properties. Note that the engine property is also an object with its own properties.

var myHonda = {color: 'red', wheels: 4, engine: {cylinders: 4, size: 2.2}};

You can also use object initializers to create arrays. See [array literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Array_literals).

Using a constructor function

Alternatively, you can create an object with these two steps:

1. Define the object type by writing a constructor function. There is a strong convention, with good reason, to use a capital initial letter.
2. Create an instance of the object with new.

To define an object type, create a function for the object type that specifies its name, properties, and methods. For example, suppose you want to create an object type for cars. You want this type of object to be called Car, and you want it to have properties for make, model, and year. To do this, you would write the following function:

function Car(make, model, year) {  
 this.make = make;  
 this.model = model;  
 this.year = year;  
}

Notice the use of this to assign values to the object's properties based on the values passed to the function.

Now you can create an object called mycar as follows:

var mycar = new Car('Eagle', 'Talon TSi', 1993);

This statement creates mycar and assigns it the specified values for its properties. Then the value of mycar.make is the string "Eagle", mycar.year is the integer 1993, and so on.

You can create any number of Car objects by calls to new. For example,

var kenscar = new Car('Nissan', '300ZX', 1992);  
var vpgscar = new Car('Mazda', 'Miata', 1990);

An object can have a property that is itself another object. For example, suppose you define an object called person as follows:

function Person(name, age, sex) {  
 this.name = name;  
 this.age = age;  
 this.sex = sex;  
}

and then instantiate two new person objects as follows:

var rand = new Person('Rand McKinnon', 33, 'M');  
var ken = new Person('Ken Jones', 39, 'M');

Then, you can rewrite the definition of Car to include an owner property that takes a person object, as follows:

function Car(make, model, year, owner) {  
 this.make = make;  
 this.model = model;  
 this.year = year;  
 this.owner = owner;  
}

To instantiate the new objects, you then use the following:

var car1 = new Car('Eagle', 'Talon TSi', 1993, rand);  
var car2 = new Car('Nissan', '300ZX', 1992, ken);

Notice that instead of passing a literal string or integer value when creating the new objects, the above statements pass the objects rand and ken as the arguments for the owners. Then if you want to find out the name of the owner of car2, you can access the following property:

car2.owner.name

Note that you can always add a property to a previously defined object. For example, the statement

car1.color = 'black';

adds a property color to car1, and assigns it a value of "black." However, this does not affect any other objects. To add the new property to all objects of the same type, you have to add the property to the definition of the Car object type.

Using the Object.create method

Objects can also be created using the [Object.create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create" \t "_blank) method. This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function.

// Animal properties and method encapsulation  
var Animal = {  
 type: 'Invertebrates', // Default value of properties  
 displayType: function() { // Method which will display type of Animal  
 console.log(this.type);  
 }  
};// Create new animal type called animal1   
var animal1 = Object.create(Animal);  
animal1.displayType(); // Output:Invertebrates// Create new animal type called Fishes  
var fish = Object.create(Animal);  
fish.type = 'Fishes';  
fish.displayType(); // Output:Fishes