**JS-Object and it’s internal representation:**

object literal is a comma-separated list of name-value pairs wrapped in curly braces.

* *var car={*

*id:1 ,*

*name:’abc’ ,*

*display:function()*

*}*

An object is a collection of properties, and a property is an association between a name (or key) and a value. A property’s value can be a function, in which case the property is known as a method.

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.

We can 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;*

The above example could also be written using an ‘object initializer’, which is a comma-delimited list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}):

* *var myCar = {  
  make: ‘Ford’,  
  model: ‘Mustang’,  
  year: 1969  
  };*

Unassigned properties of an object are [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) (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:

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

**Object constructor:**

Useful when we require to create multiple objects of similar type. In this case, a constructor (kind of blueprint) is created and multiple objects can be initialized using the new keyword using the constructor as a wrapper for the newly created objects.

Constructor function:

* *Function Person(name, age, eye){*

*this.name = name;*

*this.age = age;*

*this.eyeColor = eye;*

*}*

Creating object using constructor:

* *Var p1 = new Person(“Kalai”, 34, “blue”);*
* *Var p2 = new Person(“Selvi”, 35, “grean”);*

## Object.assign():

## It is used to copy the values and properties from one or more source objects to a target object.

## Here is an example where properties from three source objects are getting assigned to a new object using Object.assign()

* *var obj1 = { a: 10 };  
  var obj2 = { b: 20 };  
  var obj3 = { c: 30 };  
  var new\_obj = Object.assign(o1, o2, o3);  
  console.log(new\_obj);*

*Output : Object { a: 10, b: 20, c: 30 }*

## Object.fromEntries:

## This method transforms a list of key-value pairs into an object.

## *const entries = new car([ [‘id’, 4], [‘color’, ‘blue’] ]);*

## *const car1= Object.fromEntries(entries);*

## *console.log(car1); output: Object { id: 4, color: ‘blue’}*