1)http://stackoverflow.com/questions/16109108/javascript-creating-objects-multiple-approaches-any-differences

var obj = {

prop: value,

.

.

.

}

Approach one is standard approach, nothing new :)

Approach 2

var obj = new function() {

var prop1 = value1;

var fn1 = function() {

};

.

.

.

this.prop2 = value2;

.

.

.

}();

The function approach, I wanted to compare this approach with approach 3. The function approach is primarily used for encapsulation (correct?)

Approach 3

var obj = (function() {

var prop1 = value1;

var fn1 = function() {

return {

prop2: value2,

.

.

.

}

})();

2)<!DOCTYPE html>

<html>

<body>

<p>Click the button to loop through the properties of an object.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

function myFunction() {

var person = {fname:"John", lname:"Doe", age:25};

var text = "";

var x;

for (x in person) {

text += person[x] + " ";

}

document.getElementById("demo").innerHTML = text;

}

</script>

</body>

</html>

http://www.w3schools.com/jsref/jsref\_forin.asp

3)The hasOwnProperty() method returns a boolean indicating whether the object has the specified property. This method can be used to determine whether an object has the specified property as a direct property of that object; unlike the [in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/in) operator, this method does not check down the object's prototype chain.

o = new Object();

o.prop = 'exists';

function changeO() {

o.newprop = o.prop;

delete o.prop;

}

o.hasOwnProperty('prop'); // returns true

changeO();

o.hasOwnProperty('prop'); // returns false

4)The Object.keys() method returns an array of a given object's own enumerable properties, in the same order as that provided by a [for...in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) loop (the difference being that a for-in loop enumerates properties in the prototype chain as well). example:// array like object

var obj = { 0: 'a', 1: 'b', 2: 'c' };

console.log(Object.keys(obj)); // console: ['0', '1', '2']

5)The Object.defineProperties() method defines new or modifies existing properties directly on an object, returning the object.

6)Enumerable: true if and only if this property shows up during enumeration of the properties on the corresponding object.  
**Defaults to false.**

A data descriptor also has the following optional keys:

**value**

The value associated with the property. Can be any valid JavaScript value (number, object, function, etc).  
**Defaults to**[**undefined**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined)**.**

**writable**

true if and only if the value associated with the property may be changed with an [assignment operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators).  
**Defaults to false.**

**get**

A function which serves as a getter for the property, or [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) if there is no getter. The function return will be used as the value of property.  
**Defaults to**[**undefined**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined)**.**

**set**

A function which serves as a setter for the property, or [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) if there is no setter. The function will receive as only argument the new value being assigned to the property.  
**Defaults to**[**undefined**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined)**.**

7)The**new operator** creates an instance of a user-defined object type or of one of the built-in object types that has a constructor function

You can create any number of car objects by calls to new. For example:

var kenscar = new Car("Nissan", "300ZX", 1992);