**Objects and its Internal Representation in JavaScript**

**Introduction:**

JavaScript is an object-oriented programming language, so everything in JavaScript is an object. The Object type represents one of [JavaScript's data types](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures) and forms the building blocks for modern JavaScript.

However, an Object may be deliberately created for which this is not true (e.g. by [Object.create(null)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create)), or it may be altered so that this is no longer true (e.g. with [Object.setPrototypeOf](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/setPrototypeOf)).

These objects are quite different from JavaScript’s primitive data-types (Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types. An object, is a reference data type. “A JavaScript object is a collection of named values having state and behaviour (properties and method)”.

**For example:** Person, car, pen, bike, Personal Computer, Washing Machine etc.

**Take the case of cars.**

All cars have the same properties, but the property values differ from car to car. All cars have the same methods, but the methods are performed at different times.

**In JavaScript, almost "everything" is an object.**

* Booleans can be objects (if defined with the new keyword)
* Numbers can be objects (if defined with the new keyword)
* Strings can be objects (if defined with the new keyword)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects
* Arrays are always objects
* Functions are always objects
* Objects are always objects

All JavaScript values, except primitives, are objects.

Objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “key: value” pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

**JavaScript Primitives**

A **primitive value** is a value that has no properties or methods.

**3.14** is a primitive value

A **primitive data type** is data that has a primitive value.

JavaScript defines 7 types of primitive data types:

**Examples**

* string
* number
* boolean
* null
* undefined
* symbol
* bigint

|  |  |  |
| --- | --- | --- |
| **Value** | **Datatype** | **Comment** |
| "Hello" | string | "Hello" is always "Hello" |
| 3.14 | number | 3.14 is always 3.14 |
| true | boolean | true is always true |
| false | boolean | false is always false |
| null | null (object) | null is always null |
| undefined | undefined | undefined is always undefined |

**Immutable**

Primitive values are immutable (they are hardcoded and cannot be changed).

if x = 3.14, you can change the value of x, but you cannot change the value of 3.14.

## Objects are Variables

JavaScript variables can contain single values:

### **Example:** let person = "John Doe";

JavaScript variables can also contain many values.

Objects are variables too. But objects can contain many values.

Object values are written as **name : value** pairs (name and value separated by a colon).

A JavaScript object is a collection of named values

**Syntax:**

var <object-name> = {key1: value1, key2: value2,... keyN: valueN};

Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don’t actually store the value.

### **Example:** let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

It is a common practice to declare objects with the const keyword.

### **Example:** const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

## Object 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 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;

**Note:*Unassigned properties of an object are undefined (and not null).***

***myCar.color; // undefined***

Properties of JavaScript objects can also be accessed or set using a bracket notation. Objects are sometimes called **associative arrays**, since each property is associated with a string value that can be used to access it.

**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);

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 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

The named values, in JavaScript objects, are called **properties**.

So, simple definition for Java Script properties is “Properties are the values associated with a JavaScript object”.

The object properties can be different primitive values, other objects and functions.

Properties can usually be changed, added, and deleted, but some are read only.

Every object has some property associated with some value. These values can be accessed using these properties associated with them.

var myCar = new Object();

myCar.make = 'Suzuki';

myCar.model = 'Altros';

myCar.year = 1978;

myCar.wheels = 2;

After creating myCar object, the value inside the object can be accessed using keys.

i.e.

myCar.year

Output: 1978

These values can be accessed using brackets notation also.

myCar[year]

Output: 1978

|  |  |
| --- | --- |
| Property | Value |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |

**The syntax for adding a property to an object is:**

**ObjectName.ObjectProperty = propertyValue;**

**The syntax for deleting a property from an object is:**

**delete ObjectName.ObjectProperty;**

**The syntax to access a property from an object is:**

**objectName.property // Car.Make**

**//or**

**objectName["property”] // Car["Make"]**

**//or**

**objectName[expression] // x = "Make"; Car[x]**

## ****Object Methods****

An object method is an object property containing a function definition.

i.e.,

Let’s assume to start the car there will be a mechanical functionality.

**function(){return ignition.on}**

and so similar is to stop/brake/headlights on & off, etc.

So, Java Script Object methods is “Methods are actions that can be performed on objects.”.

## Create a JavaScript Object

There are three ways in which we can create a JavaScript object. Let’s go through each method:

## Create JavaScript Object with Object Literal

One of easiest way to create a javascript object is object literal, simply define the property and values inside curly braces as shown below

let bike = {name: 'SuperSport', maker:'Ducati', engine:'937cc'};

## Create JavaScript Object with Constructor

Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavor as shown below

function Vehicle(name, maker) {

this.name = name;

this.maker = maker;

}

let car1 = new Vehicle(’Fiesta’, 'Ford’);

let car2 = new Vehicle(’Santa Fe’, 'Hyundai’)

console.log(car1.name); //Output: Fiesta

console.log(car2.name); //Output: Santa Fe

We can also use an object constructor to initialize a JavaScript object. This method is also commonly known as **object prototyping**.

The Object constructor's behaviour depends on the input's type.

* If the value is [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null) or [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined), it will create and return an empty object.
* If the value is an object already, it will return the value.
* Otherwise, it will return an object of a Type that corresponds to the given value.

***When called in a non-constructor context, Object behaves identically to new Object().***

## Using the JavaScript Keyword new

We can use the new [keyword](https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-this-keyword) to create and define an object. The following example also creates a new JavaScript object with four properties:

var person = new Object();

person.firstName = “John”;

person.lastName = “Doe”;

person.age = 50;

person.eyeColor = “blue”;

## Using the Object.create method

Objects can also be created using the Object.create() 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