**4 WAYS TO CREATE AN OBJECT IN JAVASCRIPT (WITH EXAMPLES)**

**BY**[**DEVELOPERDRIVE STAFF**](https://www.developerdrive.com/author/DeveloperDrive-Staff)

JavaScript is a modern object-oriented programming language. It has been designed as a set of objects that interact with each other. Object-oriented languages such as JavaScript, C++, or Ruby address the shortfalls of traditional procedural languages such as C or Pascal that focus on actions and procedures instead of objects. In JavaScript, you can make objects in a number of different ways. In this guide, you’ll learn step by step how you can create new JavaScript objects.

**WHAT IS A JAVASCRIPT OBJECT?**

A JavaScript object is a variable that can hold many different values. It acts as the container of a set of related values. For example, users of a website, payments in a bank account, or recipes in a cookbook could all be JavaScript objects.

In JavaScript, objects can store two kinds of values:

1. *properties* for static values
2. *methods* for dynamic values

When you create a JavaScript object, you need to define its name, properties, and methods.

**CREATE A JAVASCRIPT OBJECT**

You can make a JavaScript object in four different ways:

1. with object literals
2. using a constructor function
3. with ECMAScript 6 classes
4. with the *Object.create()* method

Let’s see them one by one below.

**1. OBJECT LITERALS**

Defining an object literal is the simplest way to create a JavaScript object. As objects are variables, you can instantiate them the same way as a variable. For example, the following code creates an object called *user001* with three properties: *firstName*, *lastName,* and *dateOfBirth*:

  var user001 = {

firstName: "John",

lastName: "Smith",

dateOfBirth: 1985

};

If you open your console in your web browser you can use the [console.log()](https://developer.mozilla.org/en-US/docs/Web/API/Console/log) function to test if the object has really been created:

console.log(user001);

// {firstName: "John", lastName: "Smith", dateOfBirth: 1985}

You can also check each property separately by calling the property names, using a simple dot notation:

console.log(user001.dateOfBirth);

// 1985

You can also add a method to an object literal. For example, the *getName()* method below takes two properties of the *user001* object (*firstName* and *lastName*) and returns the user’s full name. The [*this* keyword](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) refers to the current object of which properties the method is calling.

var user001 = {

firstName: "John",

lastName: "Smith",

dateOfBirth: 1985,

getName: function(){

return "User's name: " + this.firstName + " " + this.lastName;

}

};

You can check the *getName()* method in the console using the same dot notation. However, don’t forget to put parentheses after the name of the method, as this is how JavaScript differentiates methods from properties. If you leave out the parentheses the console won’t execute the method, as it will be looking for a property called *getName*instead of the method called *getName()*.

console.log(user001.getName());

// User's name: John Smith

You can’t only define simple values for properties. It’s also possible to use objects as properties of objects. This feature is pretty useful when you want to structure the data your object stores. Below, the *user001* object holds the *spokenLanguages* property that’s also an object. You can see that it’s defined exactly the same way as any other object literal.

var user001 = {

firstName: "John",

lastName: "Smith",

dateOfBirth: 1985,

spokenLanguages: {

native: "English",

fluent: "Spanish",

intermediate: "Chinese"

}

};

Now, when you are printing out the value of the *spokenLanguages* property, the console returns the whole object.

console.log(user001.spokenLanguages);

// {native: "English", fluent: "Spanish", intermediate: "Chinese"}

However, you can also print out just one property of *spokenLanguages*, using the same dot notation:

console.log(user001.spokenLanguages.intermediate);

// Chinese

Besides objects, you can also use arrays as object properties. This is especially useful when you don’t want to define the property as key-value pairs, just as a simple list of values. The following code creates the same *spokenLanguages* property as before, but as an array:

var user001 = {

firstName: "John",

lastName: "Smith",

dateOfBirth: 1985,

spokenLanguages: ["English", "Spanish", "Chinese"]

};

When you now check the value of the property, the console will return it as an array. Defining a property as an array (as opposed to an object) has another advantage. You can quickly find out the number of its elements by calling the [*length*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/length) property of the built-in Array() object.

console.log(user001.spokenLanguages);

// (3) ["English", "Spanish", "Chinese"]

console.log(user001.spokenLanguages.length);

// 3

Object literals are the instances of JavaScript’s global [*Object()*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) object type. JavaScript has a number of [built-in objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects) such as *Object()* and *Array()*that have their own pre-defined properties and methods you can call on them. For instance, the aforementioned *length*property of the *Array()*object is such as a pre-defined property.

**2. CONSTRUCTOR FUNCTIONS**

The second method of creating a JavaScript object is using a constructor function. As opposed to object literals, here, you define an object type without any specific values. Then, you create new object instances and populate each of them with different values.

Below, you can see the same *user001* object defined by using a constructor function called *function User().* The constructor creates an object type called *User().* Then, we create a new object instance called *user001,* using the [*new*operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new). The constructor function contains three *this* statements that define the three properties with empty values. The values of the properties are added by each object instance.

function User(firstName, lastName, dateOfBirth) {

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = dateOfBirth;

}

var user001 = new User("John", "Smith", 1985);

The console returns the *user001* object the same way as before. However, this time it’s the instance of the custom *User()* object type instead of the pre-built*Object().*This is the main thing in which object literals and objects created with constructors are different from each other.

console.log(user001);

// User {firstName: "John", lastName: "Smith", dateOfBirth: 1985}

Besides properties, you can also define methods within a constructor function. You need to use almost the same syntax as with methods created for object literals. The only difference is that here, you also need to add the *this* keyword before the name of the method.

function User(firstName, lastName, dateOfBirth) {

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = dateOfBirth;

this.getName = function(){

return "User's name: " + this.firstName + " " + this.lastName;

}

}

var user001 = new User("John", "Smith", 1985);

When you test the method in the console, it returns the same result as before. Here, also don’t forget to put parentheses after the method’s name.

console.log(user001.getName());

// User's name: John Smith

As I mentioned before, JavaScript has a number of pre-built object types you can initialize with the *new* keyword. You can do that because JavaScript has pre-made constructors for these objects, so you don’t have to define them by yourself.

For example, the code below creates a new instance of the *Date()* global object. If you take a look at the [docs](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date) you’ll see that JavaScript defines four different constructors for the *Date()* object (the four *new*statements). You can use any of them. You should choose the best for your needs. The *today* object below uses the first constructor of the *Date()* object type; the one that doesn’t take any arguments and returns the current date.

var today = new Date();

console.log(today);

// Wed Nov 14 2018 08:52:43 GMT+0100

**3. ECMASCRIPT 6 CLASSES**

ECMAScript 6 introduced a new syntax for creating a JavaScript object—the [class syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes). Although JavaScript is an object-oriented language, before ES6, it didn’t use classes as other OOPs languages like Java do. The new class syntax doesn’t add any new logic to JavaScript; it’s basically nothing more than syntactical sugar. But, it’s a nice feature, especially if you are coming from another OOPs language and missing the good ol’ class syntax.

With the new ES6 class syntax, the *user001* object can be created in the following way:

class User {

constructor(firstName, lastName, dateOfBirth) {

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = dateOfBirth;

this.getName = function(){

return "User's name: " + this.firstName + " " + this.lastName;

}

}

}

var user001 = new User("John", "Smith", 1985);

The *user001* object will be an instance of the custom *User()* class, just like when it was created with the traditional constructor syntax.

**4. THE OBJECT.CREATE() METHOD**

The last (but not the least) way to create a JavaScript object is using the [*Object.create()*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create) method. It’s a standard method of JavaScript’s pre-built *Object*object type. The *Object.create()* method allows you to use an existing object literal as the prototype of a new object you create.

Say, you want to create a *user002* object that has the same properties and methods as *user001*, just with different values*.* I copied below the declaration of *user001* but the interesting part starts at the declaration of *user002*.

You use the *Object.create()* method to instantiate the new *user002* object. You need to add *user001* as an argument of the *create()* method, as that will the prototype of the new object. Then, you simply set the values for the three properties (*firstName, lastName, dateOfBirth*) using the familiar dot notation.

var user001 = {

firstName: "John",

lastName: "Smith",

dateOfBirth: 1985,

getName: function(){

return "User's name: " + this.firstName + " " + this.lastName;

}

};

var user002 = Object.create(user001);

user002.firstName = "Jane";

user002.lastName = "King";

user002.dateOfBirth = 1989;

When you test the new *user002* object in the console, you’ll see that it has been populated with the new values:

console.log(user002);

// {firstName: "Jane", lastName: "King", dateOfBirth: 1989}

console.log(user002.dateOfBirth);

// 1989

console.log(user002.getName());

// User's name: Jane King

The objects you create with the *Object.create()* method are also object literals and the instances of JavaScript’s built-in *Object()* object type.

**WRAPPING UP & NEXT STEPS**

In JavaScript, you can use four different techniques to create new objects.

You can create an object literal (with other words a standalone object) by either defining its properties and methods or using the *Object.create()*method. These objects are the instances of the global *Object()*object type.

Or, you can use a constructor defined with either the traditional or the new ES6 class syntax to create a custom object type, for example *User()*. This object type will be the parent of a number of object instances, for example *user001*and *user002*.

That’s all about creating JavaScript objects. If you want to read more about JavaScript techniques, don’t miss out our collection of [experimental JavaScript projects](http://www.developerdrive.com/2018/02/11-experimental-javascript-projects-pushing-the-boundaries-of-scripting/), either. And, if you are interested in learning front-end development we have a [fresh collection of learning resources](http://www.developerdrive.com/2018/02/18-learning-resources-for-front-end-developers/), too.