# Creating objects in JavaScript

JavaScript is an object-based language based on prototypes, rather than being class-based. Because of this different basis, it can be less apparent how JavaScript allows you to create hierarchies of objects and to have an inheritance of properties and their values.

## 1) Creating objects using object literal syntax

This is really simple. All you have to do is throw your key value pairs separated by ':' inside a set of curly braces({ }) and your object is ready to be served (or consumed), like below:

```
const person = {
  firstName: 'testFirstName',
  lastName: 'testLastName'
};
```

## 2) Creating objects using the 'new' keyword

This method of object creation resembles the way objects are created in class-based languages, like Java. By the way, starting with ES6, classes are native to JavaScript as well and we will look at creating objects by defining classes towards the end of this article. So, to create an object using the 'new' keyword, you need to have a constructor function.

**Here are 2 ways you can use the 'new' keyword pattern —**

### a) Using the 'new' keyword with' in-built Object constructor function

To create an object, use the new keyword with Object() constructor, like this:

```
const person = new Object();
```

Now, to add properties to this object, we have to do something like this:

```
person.firstName = 'testFirstName';
person.lastName = 'testLastName';
```

### b) Using 'new' with user defined constructor function

The other problem with the approach of using the 'Object' constructor function result from the fact that every time we create an object, we have to manually add the properties to the created object. So, to get rid of manually adding properties to the objects, we create custom (or user-defined) functions. We first create a constructor function and then use the 'new' keyword to get objects:

```
function Person(fname, lname) {
  this.firstName = fname;
  this.lastName = lname;
}
```

Now, anytime you want a 'Person' object, just do this:

```
const personOne = new Person('testFirstNameOne', 'testLastNameOne');
const personTwo = new Person('testFirstNameTwo', 'testLastNameTwo');
```

# 3) Using Object.create() to create new objects

The **Object.create()** method creates a new object, using an existing object as the prototype of the newly created object. It takes two parameters. The first parameter is a mandatory object that serves as the prototype of the new object to be created. The second parameter is an optional object which contains the properties to be added to the new object.

Imagine you have an organization represented by *orgObject*

```
const orgObject = { company: 'ABC Corp' };
```

And you want to create employees for this organization. Clearly, you want all the employee objects.

```
const employee = Object.create(orgObject, { name: { value: 'EmployeeOne' } });

console.log(employee); // { company: "ABC Corp" }

console.log(employee.name); // "EmployeeOne"
```

# 4) Using Object.assign() to create new objects

The Object**.**assign**()** method is used to copy the values of all enumerable own properties from one or more source objects to a target object. It will return the target object. Object.assign method can take any number of objects as parameters. The first parameter is the object that it will create and return. The rest of the objects passed to it will be used to copy the properties into the new object.
Assume you have two objects as below:

```
const orgObject = { company: 'ABC Corp' }

const carObject = { carName: 'Ford' }
```

Now, you want an employee object of 'ABC Corp' who drives a 'Ford' car. You can do that with the help of Object.assign as below:

```
const employee = Object.assign({}, orgObject, carObject);
```

Now, you get an `employee` object that has `company` and `carName` as its property.

```
console.log(employee); // { carName: "Ford", company: "ABC Corp" }
```

# 5) Using ES6 classes to create objects

This method is similar to using 'new' with user defined constructor function. The constructor functions are now replaced by classes as they are supported through ES6 specifications.

```
class Person {
  constructor(fname, lname) {
    this.firstName = fname;
    this.lastName = lname;
  }
}


const person = new Person('testFirstName', 'testLastName');


console.log(person.firstName); // testFirstName

console.log(person.lastName); // testLastName
```