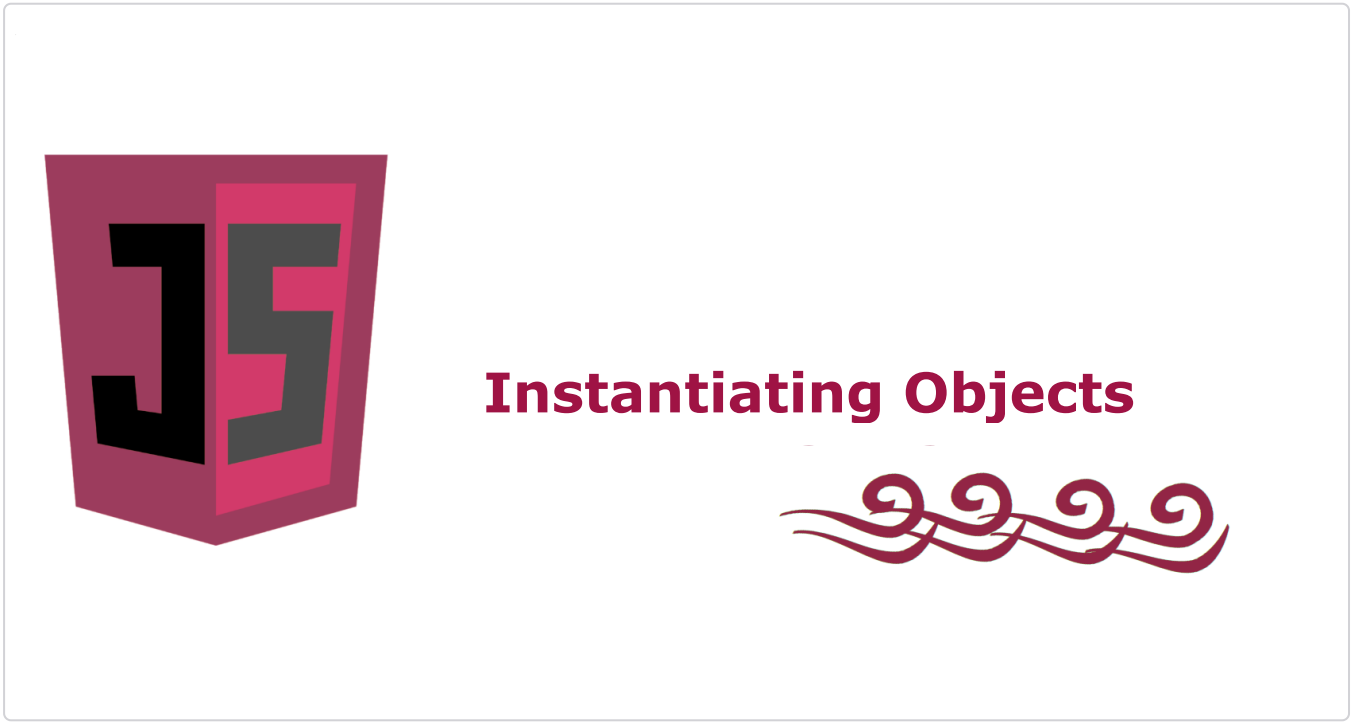
Instantiating Objects

In this lesson, we instantiate objects in JavaScript!
Let's begin!

WE'LL COVER THE FOLLOWING

- Listing 8-14: Constructor function with method
- Listing 8-15: Constructor function with global method



In most programming languages, constructors are the only way to instantiate new objects. As you already saw in the introduction of this section, JavaScript allows you to use manual setup and you can create objects using the object literal (JSON). The most concise way is by using the object literal; it is easy to read and helps you use less characters when typing.

However, it has a significant issue. You can only instantiate your objects one by one, and you must repeat the literal every time, often including dozens of lines. This way is very error-prone and pretty laborious to maintain.

JavaScript provides many ways to instantiate objects, and several construction

JavaScript provides many ways to instantiate objects, and several construction patterns have been applied by JavaScript experts. Most patterns mix generic creation patterns with the peculiarities of JavaScript object management. In this section you will learn a few of the most frequently used patterns and their pros and cons.

First, let's see the standard constructor pattern. You already know this pattern, so Listing 8-14 will look familiar to you:

Listing 8-14: Constructor function with method

```
<!DOCTYPE html>
<html>
<head>
  <title>Constructor function with method</title>
  <script>
    var Employee = function (id, firstName,
      lastName, title) {
      this.id = id;
      this.firstName = firstName;
      this.lastName = lastName;
      this.title = title;
      this.getFullName = function () {
        return this.lastName + ", " + this.firstName;
      }
    }

    var philip = new Employee(1, "Philip", "Moore",
      "CEO");
    var jane = new Employee(2, "Jane", "Mortimer",
      "CFO");

    console.log(philip.getFullName());
    console.log(jane.getFullName());
  </script>
</head>
<body>
  Listing 8-14: View the console output
</body>
</html>
```

At first sight, this constructor function seems perfect. It assigns the initial values of properties and creates the `getFullName()` method. You can easily instantiate an Employee object by using the new operator with the `Employee()` constructor function.

Although it looks perfect, it conceals an important issue. Every time you create a new Employee instance it creates a new `getFullName()` method that is an exact copy of the one in the constructor function. When you create 10,000

instances, you'll get 10,000 copies of `getFullName()`, and it is a waste of resources.

Extracting the `getFullName()` method from the object's scope solves this issue, as shown in Listing 8-15.

Listing 8-15: Constructor function with global method

```
<!DOCTYPE html>
<html>
<head>
  <title>Constructor function with global method</title>
  <script>
    var Employee = function (id, firstName,
      lastName, title) {
      this.id = id;
      this.firstName = firstName;
      this.lastName = lastName;
      this.title = title;
      this.getFullName = getEmployeeFullName;
    }

    function getEmployeeFullName() {
      return this.lastName + ", " + this.firstName;
    }

    var philip = new Employee(1, "Philip", "Moore",
      "CEO");
    var jane = new Employee(2, "Jane", "Mortimer",
      "CFO");

    console.log(philip.getFullName());
    console.log(jane.getFullName());
  </script>
</head>
<body>
  Listing 8-15: View the console output
</body>
</html>
```

This solution eliminates the issue with multiple method instances. Setting the instance's `getFullName` property to `getEmployeeFullName` simply assigns a function pointer to the property but does not create an individual method instance. This creates a new problem.

If you create constructor functions for objects that have a large number of methods, the global scope will be scattered with a number of functions that belong to certain object types. This hurts the principle of encapsulation (an object should encapsulate its data and operations) and makes your code less maintainable.

maintainable.

In the *next lesson*, we will learn to use object prototypes in our JavaScript program.

See you there!