Objects and Dynamism

In this lesson, we'll get acquainted with objects and their usage in JavaScript.
Let's begin! :)

WE'LL COVER THE FOLLOWING

In previous code examples you could already see that JavaScript works with **primitive types**, such as numbers or strings.

However, when you worked with the DOM in the previous chapter, you saw that the language works with objects representing the document loaded into the browser or a collection of objects that represent HTML elements.

Objects are the **most important concept in the language**.

Everything is either an object in JavaScript, or a primitive *type's instance that can work like an object.*

Objects can have zero, one, or more properties with their associated values, and— wait for it! — **they can be extended with properties dynamically at run-time**.

Listing 7-1 demonstrates this concept:

Listing 7-1: Extending object properties dynamically

```
{
  "name": "unraveling-html5-2nd-edition",
  "version": "1.0.0",
  "description": "Code samples for the Unraveling HTML5, CSS, and JavaScript book",
  "scripts": {
    "start": "live-server --port=8080"
  },
  "keywords": [],
  "author": "Istvan Novak (dotneteer@hotmail.com)",
  "license": "ISC"
}
```

The script in this listing produces the following console output:

console

```
Object {manufacturer: "Honda",
  type: "FR-V", regno: "ABC-123"}
Object {manufacturer: "Honda",
  type: "FR-V", regno: "ABC-123",
  year: 2007}
```

JS console Output

Show Useful Info

How did this short code create the output?

⥱ When you initiated the car object with `new Object()`, it had only a few predefined properties that each object has by default.

⥱ Setting explicit properties such as manufacturer, type, and regno (using the dot (".") notation that is very common in many programming languages) gave meaning to the car object.

⥱ Logging the object to the output displayed the content of car in a syntax called JSON (JavaScript Object Notation).

⥱ As you can see from the output, setting the year property immediately added it to the other properties held by car.

When you design software based on objects, you generally assign behavior to the object.

In most programming languages with object-oriented features, like in C++, C#, and Java, you define the common behavior in the source code and specialize objects by inheriting new types from existing ones.

In contrast to this approach, JavaScript uses constructors as templates to create new instances of objects.

The above is shown in Listing 7-2 below:

Listing 7-2: Using constructors as templates

```
<!DOCTYPE html>
<html>
<head>
  <title>Object constructor</title>
  <script>
    // Define a constructor for Car
    var Car = function (manuf, type, regno) {
      this.manufacturer = manuf;
      this.type = type;
      this.regno = regno;
    }

    // Create a Car
    var car = new Car("Honda",
      "FR-V", "ABC-123");

    // Display car properties
    console.log(car);
    // Oh, I forgot the year...
    car.year = 2007;

    // Now, display it again
    console.log(car);
  </script>
</head>
<body>
  Listing 7-2: View the console output
</body>
</html>
```

> 📜**NOTE:** In this listing I omitted all wrapper HTML markup for the sake of brevity. In this chapter's source code download you will find the full markup. From now on, I will omit HTML markup from listings whenever it does not lead to ambiguity.

The constructor is defined as a function named `Car`. When you create a `new Car()` instance, this constructor function is called with the `new` keyword. This instructs the JavaScript engine to create an object in the **memory** and pass it to the constructor function.

In this case, the constructor function uses the `this` keyword as a reference to the newly created object and it can be used to set up object instance properties.

Although the constructor function does not retrieve any value *explicitly*, the JavaScript engine *implicitly* takes the object referenced by this as the return value.

> 📜**NOTE:** JavaScript allows you to call the constructor function without the `new` operator, but in this case, it has a totally different behavior. You will learn about it later.

When you instantiate a new object with a constructor function, the JavaScript engine creates an empty *"bag"* and passes it to the constructor.

Using the `this` operator, you can put properties into this bag, and the constructor function returns a bag full of the properties you put in.

As Listing 7-2 above shows, you can put more properties into this bag at any time.

This listing's output is very similar to the output of Listing 7-1:

console

```
Car {manufacturer: "Honda",
  type: "FR-V", regno: "ABC-123"}
Car {manufacturer: "Honda",
  type: "FR-V", regno: "ABC-123",
  year: 2007}
```

JS console Output

The only difference in this new output is that `Car` is written as *the name of the logged object*, whereas previously it was Object.

Right now, I'm not going to explain this; but later, when you know more about types, I'll return to a full explanation.

In the *next lesson*, let's learn about Functions in JavaScript.