COMP30680
Web Application Development

JavaScript part 3 - Objects

David Coyle
d.coyle@ucd.ie

JavaScript Objects

In JavaScript, almost "everything" is an object.

- Booleans can be objects (or primitive data treated as objects)
- Numbers can be objects (or primitive data treated as objects)
- Strings can be objects (or primitive data treated as objects)
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are objects

Built-in JavaScript Constructors

JavaScript has built-in constructors for native objects:

```
var x1 = new Object();    // A new Object object
var x2 = new String();    // A new String object
var x3 = new Number();    // A new Number object
var x4 = new Boolean()    // A new Boolean object
var x5 = new Array();     // A new Array object
var x6 = new RegExp();    // A new RegExp object
var x7 = new Function();  // A new Function object
var x8 = new Date();      // A new Date object
```

There is generally no reason to create complex objects.

- Primitive values execute much faster, than string or number objects.
- And there is no reason to use new Array(). Use array literals instead: []
- And there is no reason to use new Function(). Use function expressions instead: function () {}.
- And there is no reason to use new Object(). Use object literals instead: {}

Built-in JavaScript Constructors

JavaScript has built-in constructors for native objects:

```
var x1 = new Object();     // A new Object object
var x2 = new String();     // A new String obj
var x3 = new Number();     // A new Number obj
var x4 = new Boolean()     // A new Boolean ob
var x5 = new Array();      // A new Array obje
var x6 = new RegExp();     // A new RegExp obj
var x7 = new Function();   // A new Function o
var x8 = new Date();       // A new Date objec
```

```
var x1 = {};              // new object
var x2 = "";              // new primitive string
var x3 = 0;               // new primitive number
var x4 = false;           // new primitive boolean
var x5 = [];              // new array object
var x6 = /()/             // new regexp object
var x7 = function(){};    // new function object
```

There is generally no reason to create complex objects.

- Primitive values execute much faster, than string or number objects.
- And there is no reason to use new Array(). Use array literals instead: []
- And there is no reason to use new Function(). Use function expressions instead: function () {}.
- And there is no reason to use new Object(). Use object literals instead: {}

JavaScript Data Types: Objects

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

JavaScript objects are containers for **named values**.

JavaScript objects are written with curly braces.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Object named values are written as **name : value** pairs, separated by commas.
These name : values pairs in JavaScript objects are called **properties**.

The object (person) has 4 properties: firstName, lastName, age, and eyeColor.

Spaces and line breaks are not important.
These two examples on this page do the same thing.

```
var person = {
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue"
};
```

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

1. Define and create a single object, using an object literal.
2. Define and create a single object, with the keyword new.
3. Define an object constructor, and then create objects of the constructed type.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

1. **Define and create a single object, using an object literal.**
2. Define and create a single object, with the keyword new.
3. Define an object constructor, and then create objects of the constructed type.

Using an object literal, you both define and create an object in one statement.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Using the JavaScript keyword **new**

The following example also creates a new JavaScript object with four properties, using the **new** keyword:

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

This does the same as the previous slide.

For simplicity, readability and execution speed, it is better to used the object literal method when creating single objects. There is generally no need to use new Object().

Using an Object Constructor

The previous examples create a single object.

An **object constructor function** can be used to create many objects of one type.

```
function person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```

The function “person” is an object constructor.

Once you have an object constructor, you can create many new objects of the same type.

Object Properties

The name : values pairs in JavaScript objects are called **properties**. A JavaScript object is a collection of unordered properties.

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

Properties can be accessed in a number of ways:

```
objectName.property          // person.age
```

```
objectName["property"]       // person["age"]
```

```
objectName[expression]       // x = "age"; person[x]
```

For further details on Object Properties see: http://www.w3schools.com/js/js_properties.asp

Manipulating Properties

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

You can **add** new properties to an *existing* object by giving it a value:

```
person.nationality = "English";
```

The **delete** keyword deletes a property from an object:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
delete person.age;   // or delete person["age"];
```

Note: the delete keyword deletes both the value of the property and the property itself.

Object Prototypes

Every JavaScript object has a prototype.

All JavaScript objects inherit their properties and methods from their prototype.

The **Object.prototype** is on the top of the prototype chain. Ultimately all JavaScript objects inherit from Object.prototype.

- Objects created using an object literal, or with new Object(), inherit directly from a prototype called Object.prototype.
- Other objects inherit indirectly: e.g. objects created with new Date() inherit the Date.prototype, which in turn inherits from the Object.prototype.

Object Prototypes

The standard way to create an new object prototype is to use an object constructor function:

```
function person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
```

You can then use the **new** keyword to create new objects from the same prototype:

```
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```

Prototypes vs individual objects

Once you have created prototypes objects and then individual instances of those objects, you can manipulate either the prototype or the individual object:

```
function person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
```

```
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```

For example adding a property to an object:

```
myFather.nationality = "English";
```

```
person.nationality = "English";
```

```
person.prototype.nationality = "English";
```

The property will be added to myFather. Not to myMother. Not to any other person objects.

This will not work. Person is an object prototype, not an actual object.

The JavaScript prototype property also allows you to add new methods to an existing prototype

Object Methods

Methods are **actions** that can be performed on objects.

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

You create an object method with the following syntax:

```
methodName : function() { code lines }
```

```
12  <script>
13  var person = {
14      firstName: "John",
15      lastName : "Doe",
16      id       : 5566,
17      fullName : function() {
18         return this.firstName + "
19      }
20  };
21
```

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Accessing Object Methods

```
methodName : function() { code lines }
```

Once created you access an object method with the following syntax:

```
objectName.methodName()
```

This example accesses the fullName() **method** of a person object:

```
name = person.fullName();
```

See **method_example.html**

Further reading

Please read the tutorials from w3schools for further details and examples of objects:

Objects: http://www.w3schools.com/js/js_object_definition.asp (See all 4 pages here.)

Questions, Suggestions?

Next:

Working with JSON data and RESTful APIs.