# Preface

This preface will include a short section on getting Node running on a computer and the object model used in the book.

# Object-Based Programming in JavaScript

Modern data structures are implemented as objects. Because of this, we need to understand how to use objects in JavaScript. JavaScript's object model is different than languages such as C++ and Java. JavaScript is a prototype-based language rather than a class-based language, which is the case for C++ and Java, to mention just two classical object-oriented programming (OOP) languages. This chapter provides an overview of how objects work in JavaScript and how we will use objects throughout the rest of the book.

## JavaScript Objects

There are three things we need to know about objects in JavaScript: 1) how to create them; 2) how to modify them; and 3) how to retrieve the data stored in them. There are several ways to perform all three of these tasks.

### Creating and Using Object Literals

The first way to create an object is by creating an object literal. An object literal consists of a pair of curly braces surrounding a set of name/value pairs. For example:

var david = {

"first\_name": "David",

"last\_name": "Durr"

};

This object has two properties, "first\_name" and "last\_name". Strictly speaking, the quotes around the property names are not required since first\_name and last\_name are legal JavaScript identifiers, so the code above can be rewritten as:

var david = {

first\_name: "David",

last\_name: "Durr"

};

After the property name, a colon separates the property name from the associated value. Object values must be entered as legal JavaScript types, so in the code above the values must be written with quotes since they are strings. After each value a comma is placed to indicate the beginning of a new property/value pair, except when the last value is written. The object definition is closed by writing a closing curly brace followed by a semicolon.

An object's properties can consist of any legal JavaScript object, so arrays and other objects can be used as properties. Object values can be of any legal JavaScript data type. Here is another object literal using more advanced types and data:

var student1 = {

id: "1234",

courses: ["Programming I", "English II", "Algebra"],

advised: false

};

Retrieving the values of an object's properties can be achieved in two ways. A value can be referenced from an object using the dot operator ( . ), or the property name can be written as a string inside a [ ] suffix. Here are examples of both, using the objects from above:

console.log(david.last\_name); // displays "Durr"

console.log(student1["id"]); // displays "1234"

One of the values from the courses property of the student1 object can be retrieved using standard array notation:

console.log(student1.courses[1]); displays "English II"

Object literals are updated using standard assignment:

student1.advised = true;

student1.courses[1] = "English Composition II";

If you try to update an object's property that doesn't exist, the property is added to the list of the object's properties:

david.age = 50;

### Creating Objects Using Constructor Functions

An alternative way to create objects in JavaScript is to use a constructor function to create a general object and then use the new keyword to instantiate an instance of the object. This technique has much the same feel for how class objects are instantiated in Java and C# and will be used throughout this book when objects are used.

To demonstrate how objects are created using constructor functions, we'll create a Person object with several properties and a couple of methods. A constructor function in JavaScript looks like a regular function, for example:

function Person(first, middle, last) {

this.first = first;

this.middle = middle;

this.last = last;

}

We start with the keyword function followed by the name of the object. You'll notice the object name is capitalized. This convention is not required by JavaScript but, because it is a standard convention in most OOP languages, we will use it here.

The object name is followed by a parameter list used to provide values for the object's properties. In this example, our Person object will have a first, middle, and last name, so those we provide three parameters for that data.

When assigning data to the object's properties, we have to use the keyword this to reference the current, or working, object. You may be accustomed to having the option of using or not using this in constructor functions in other languages, but you must use it in JavaScript constructor functions. In fact, as we'll see later, you must use this anytime you reference any property or method of an object in a function.

Once we've created a constructor function for an object, we can use the function to create instances of an object:

var aPerson = new Person("John","Quincy","Public");

Once the object is created, we have full access to the properties of the main object in our instance:

console.log("First name: " + aPerson.first);

console.log("Middle name: " + aPerson.middle);

console.log("Last name: " + aPerson.last);

We can write methods for the object by defining a function and adding the name of the function to the constructor function:

function Person(first, middle, last) {

this.first = first;

this.middle = middle;

this.last = last;

this.initials = initials;

}

function initials() {

return this.first[0] + this.middle[0] + this.last[0];

}

var aPerson = new Person("John","Quincy","Public");

console.log("First name: " + aPerson.first);

console.log("Middle name: " + aPerson.middle);

console.log("Last name: " + aPerson.last);

console.log("Initials: " + aPerson.initials());

The output from this program is:

First name: John

Middle name: Quincy

Last name: Public

Initials: JQP