CHAPTER 1

The JavaScript Programming Environment and Model

This chapter describes the JavaScript programming environment and the programming constructs we'll use in this book to define the various data structures and algorithms examined.

The JavaScript Environment

JavaScript has historically been a programming language that ran only inside a web browser. However, in the past few years, there has been the development of JavaScript programming environments that can be run from the desktop, or similarly, from a server. In this book we use one such environment: the JavaScript shell that is part of Mozilla's comprehensive JavaScript environment known as SpiderMonkey.

To download the JavaScript shell, navigate to the Nightly Build web page. Scroll to the bottom of the page and pick the download that matches your computer system.

Once you've downloaded the program, you have two choices for using the shell. You can use it either in interactive mode or to interpret JavaScript programs stored in a file. To use the shell in interactive mode, type the command `js` at a command prompt. The shell prompt, `js>`, will appear and you are ready to start entering JavaScript expressions and statements.

The following is a typical interaction with the shell:

```
js> 1
1
js> 1+2
3
js> var num = 1;
js> num*124
124
```

```
js> for (var i = 1; i < 6; ++i) {
    print(i);
}
1
2
3
4
5
js>
```

You can enter arithmetic expressions and the shell will immediately evaluate them. You can write any legal JavaScript statement and the shell will immediately evaluate it as well. The interactive shell is great for exploring JavaScript statements to discover how they work. To leave the shell when you are finished, type the command `quit()`.

The other way to use the shell is to have it interpret complete JavaScript programs. This is how we will use the shell throughout the rest of the book.

To use the shell to intepret programs, you first have to create a file that contains a JavaScript program. You can use any text editor, making sure you save the file as plain text. The only requirement is that the file must have a *.js* extension. The shell has to see this extension to know the file is a JavaScript program.

Once you have your file saved, you interpret it by typing the `js` command followed by the full filename of your program. For example, if you saved the `for` loop code fragment that's shown earlier in a file named *loop.js*, you would enter the following:

```
c:\js>js loop.js
```

which would produce the following output:

```
1
2
3
4
5
```

After the program is executed, control is returned to the command prompt.

JavaScript Programming Practices

In this section we discuss how we use JavaScript. We realize that programmers have different styles and practices when it comes to writing programs, and we want to describe ours here at the beginning of the book so that you'll understand the more complex code we present in the rest of the book. This isn't a tutorial on using JavaScript but is just a guide to how we use the fundamental constructs of the language.

Declaring and Intializing Variables

JavaScript variables are global by default and, strictly speaking, don't have to be declared before using. When a JavaScript variable is initialized without first being declared, it becomes a global variable. In this book, however, we follow the convention used with compiled languages such as C++ and Java by declaring all variables before their first use. The added benefit to doing this is that declared variables are created as local variables. We will talk more about variable scope later in this chapter.

To declare a variable in JavaScript, use the keyword `var` followed by a variable name, and optionally, an assignment expression. Here are some examples:

```
var number;
var name;
var rate = 1.2;
var greeting = "Hello, world!";
var flag = false;
```

Arithmetic and Math Library Functions in JavaScript

JavaScript utilizes the standard arithmetic operators:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo)

JavaScript also has a math library you can use for advanced functions such as square root, absolute value, and the trigonometric functions. The arithmetic operators follow the standard order of operations, and parentheses can be used to modify that order.

Example 1-1 shows some examples of performing arithmetic in JavaScript, as well as examples of using several of the mathematical functions.

Example 1-1. Arithmetic and math functions in JavaScript

```
var x = 3;
var y = 1.1;
print(x + y);
print(x * y);
print((x+y)*(x-y));
var z = 9;
print(Math.sqrt(z));
print(Math.abs(y/x));
```

The output from this program is:

CHAPTER 2
Arrays

The array is the most common data structure in computer programming. Every programming language includes some form of array. Because arrays are built-in, they are usually very efficient and are considered good choices for many data storage purposes. In this chapter we explore how arrays work in JavaScript and when to use them.

JavaScript Arrays Defined

The standard definition for an array is a linear collection of elements, where the elements can be accessed via indices, which are usually integers used to compute offsets. Most computer programming languages have these types of arrays. JavaScript, on the other hand, has a different type of array altogether.

A JavaScript array is actually a specialized type of JavaScript object, with the indices being property names that can be integers used to represent offsets. However, when integers are used for indices, they are converted to strings internally in order to conform to the requirements for JavaScript objects. Because JavaScript arrays are just objects, they are not quite as efficient as the arrays of other programming languages.

While JavaScript arrays are, strictly speaking, JavaScript objects, they are specialized objects categorized internally as arrays. The `Array` is one of the recognized JavaScript object types, and as such, there is a set of properties and functions you can use with arrays.

Using Arrays

Arrays in JavaScript are very flexible. There are several different ways to create arrays, access array elements, and perform tasks such as searching and sorting the elements stored in an array. JavaScript 1.5 also includes array functions that allow programmers

CHAPTER 3
Lists

Lists are one of the most common organizing tools people use in their day-to-day lives. We have to-do lists, grocery lists, top-ten lists, bottom-ten lists, and many other types. Our computer programs can also use lists, particularly if we only have a few items to store in list form. Lists are especially useful if we don't have to perform searches on the items in the list or put them into some type of sorted order. When we need to perform long searches or complex sorts, lists become less useful, especially with more complex data structures.

This chapter presents the creation of a simple list class. We start with the definition of a list abstract data type (ADT) and then demonstrate how to implement the ADT. We wrap up the chapter with some problems that are best solved with lists.

A List ADT

In order to design an ADT for a list, we have to provide a definition of the list, including its properties, as well as the operations performed on it and by it.

A list is an ordered sequence of data. Each data item stored in a list is called an *element*. In JavaScript, the elements of a list can be of any data type. There is no predetermined number of elements that can be stored in a list, though the practical limit will be the amount of memory available to the program using the list.

A list with no elements is an *empty* list. The number of elements stored in a list is called the *length* of the list. Internally, the number of elements in a list is kept in a `listSize` variable. You can *append* an element to the end of a list, or you can *insert* an element into a list after an existing element or at the beginning of a list. Elements are deleted from a list using a *remove* operation. You can also *clear* a list so that all of its current elements are removed.