



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Thinking in JavaScript

A focused guide designed to help you understand and start speaking the language of the Web

Aravind Shenoy

[PACKT]
PUBLISHING

Thinking in JavaScript

A focused guide designed to help you understand and start speaking the language of the Web

Aravind Shenoy



BIRMINGHAM - MUMBAI

Thinking in JavaScript

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: March 2014

Production Reference: 1130314

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

www.packtpub.com

Cover Image by Ankita Jha (ankitajha17@gmail.com)

Credits

Author

Aravind Shenoy

Proofreaders

Stephen Copestake

Reviewers

Arturas Lebedevas

Anirudh Prabhu

Production Coordinators

Nitesh Thakur

Pooja Chiplunkar

Content Development Editor

Neeshma Ramakrishnan

Cover Work

Prachali Bhiwandkar

Technical Editor

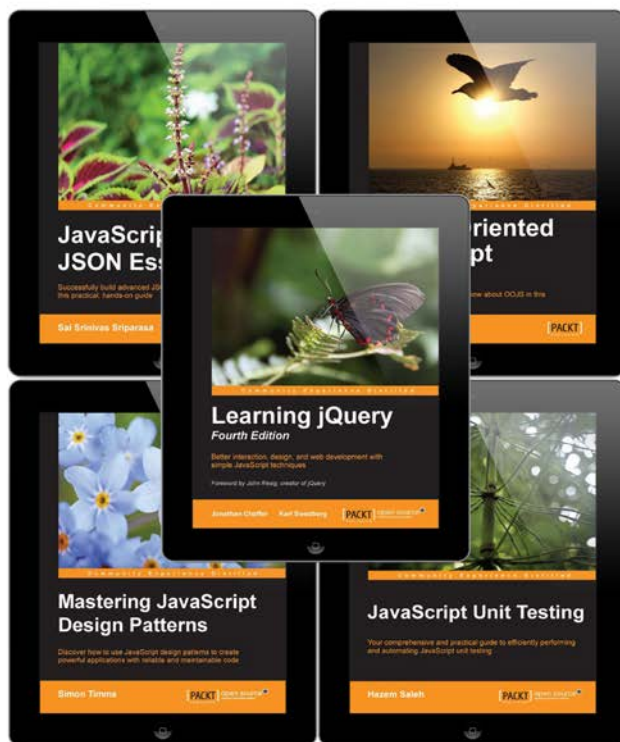
Dennis John

About the Author

Aravind Shenoy is an in-house author at Packt Publishing. An engineering graduate from the Manipal Institute of Technology, his core interests include technical writing, web designing, and software testing. He is a native of Mumbai, India, and currently resides there. He has penned down books such as *Hadoop: A Big Data Initiative* and *Thinking in CSS*. He has also authored the bestselling book: *HTML5 and CSS3 Transition, Transformation, and Animation*, Packt Publishing, (<http://www.packtpub.com/html5-and-css3-for-transition-transformation-animation/book>). He is a music buff with *The Doors*, *Oasis*, and *R.E.M* ruling his playlists.

50% OFF

YOUR NEXT EBOOK OR VIDEO



USE THE FOLLOWING CODE TO
APPLY YOUR EXCLUSIVE DISCOUNT

JAVASCRIPT50

Overview

Instead of wandering through loads of theory, we have used practical examples in this book. The examples in this book are compatible with almost every browser. Instead of using the verbatim code, you can modify the code and see the change in the output thereby understanding the subtle nuances of JavaScript.

By the end of the book, with practice, you can achieve better things and get more acquainted with JavaScript.

Thinking in JavaScript

JavaScript is a scripting language for the Web. Before we delve into the intricacies of JavaScript, we need to know why it is used. While HTML tells your browser how the page will look, JavaScript will be used for dynamic content and to add functionality.

The prerequisites for getting started with JavaScript are as follows:

- ▶ Prior knowledge of HTML
- ▶ A text editor such as Notepad
- ▶ JavaScript enabled on your web browser

JavaScript does not require a compiler to execute the code. JavaScript can be embedded within the HTML code. The web browser interprets the JavaScript while the page loads.

In this book, we are going to use a practical approach so that we can gain more insight into JavaScript. By the end of the book, we should be able to understand the concepts of JavaScript so that we can implement it in our real-life applications.

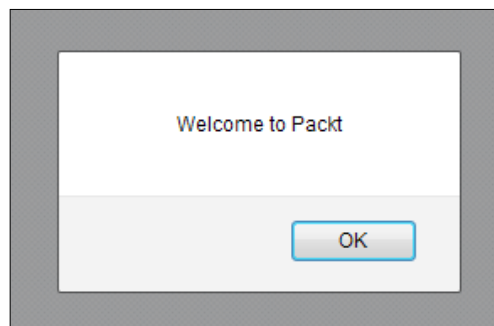
Applying JavaScript

Instead of beating around the bush and wandering through loads of theory, we will start with a simple code right away.

Let's look at the following code to understand how JavaScript is applied to a web page:

```
<!DOCTYPE html>
<html>
<head>
<title> Introduction to JavaScript </title>
<script type="text/javascript">
  alert(" Welcome to Packt ");
</script>
</head>
</html>
```

When we look at this code, we can see that we have included a line: `script type="text/javascript"` in the `<head>` section. The `script` tag and its `type` attribute tell the browser that we have introduced a JavaScript in the HTML code. We can also see that there is an `alert` command in the code. We will talk about the `alert` command later in this book. On executing the code, we can see the following output on the screen:



We can see that the JavaScript code must be inserted between the `<script>` and `</script>` tags.

Our JavaScript code can be kept in the `<head>` or `<body>` tags. It is a matter of choice and depends on the web page. If there are a lot of styles and videos, it is preferable to keep the JavaScript code in the `<body>` tag. This ensures that the web page and its resources have been loaded completely before JavaScript comes into the picture. Another reason is that any resource, except for a JavaScript file, loads asynchronously. However, when a JavaScript file is being loaded, it happens one at a time. This hampers the performance.

We must understand that if there is a lot of code to be written, it is always systematic to keep the JavaScript code separate from the HTML code. In this book, we will embed JavaScript in the HTML between the `<script>` tags for the sake of convenience. However, in real-world situations, it is good practice to keep the JavaScript code as a standalone. For that, we need to create a standalone JavaScript file and save it with a `.js` extension. The JavaScript code must be written in this separate `.js` file. Suppose we create a JavaScript file called `main.js`. We then invoke the `main.js` file in our HTML code. Let's have a look at a code example to view the procedure:

```
<!DOCTYPE html>
<html>
<head>
<title> Introduction to JavaScript </title>
<script type="text/javascript" src="main.js"></script>
</head>
</html>
```

In the `.js` file, we need to write the following code snippet:

```
alert(" Welcome to Packt ");
```

When we execute the code, we will get the same output as shown in the earlier example. As we can see, we have invoked the `main.js` file by using the following code in our HTML file:

```
<script type="text/javascript" src="main.js"></script>
```

Here, the `main.js` file should be on the same path as the HTML file. If it is in a different folder, we need to specify the entire path.

Thereby, we have seen the procedure to applying JavaScript in our code.


Variables

In JavaScript, we need containers that will store data and values for processing purposes. Hence, we use variables in JavaScript. We can assign a numeric or a string value to a variable. A string needs to be enclosed in quotes. We also have the concept of constants. A **constant** is assigned a value that is permanent in the script, unlike variables where the values can be changed.


Let's have a look at the way we declare variables and constants in JavaScript:

```
var amount;
var amount=5;
const a;
```

It is quite evident that the variable should be preceded by the `var` keyword whereas constants must be preceded by the `const` keyword. We can also see that the value can be assigned to a variable at the time of declaring it.

 `const` is compatible with modern versions of all browsers except Internet Explorer.

A variable can be either local or global. Let's look at how we differentiate between local and global variables.

 Remember that JavaScript is case-sensitive. Hence "myAttribute" is *not* the same as "myattribute". One more point to note is that every statement ends with a semicolon. Spaces inside the quotes will be reflected in the output.

Global and local variables

Global variables can be accessed from any location in the script, whereas **local variables** can be accessed only within the function.

Let's have a look at the following code to understand it better:

```
<!DOCTYPE html>
<html>
<head>
<title> Introduction to JavaScript </title>
</head>
<body>
<h1> Variables in JavaScript </h1>
<script type="text/javascript">
    function subtraction(a, b)
    {
        var c;
        c = a-b;
        return c;
    }
    var z = subtraction(10, 7);
    document.write("After subtraction, the difference is " + z);
</script>
</body>
</html>
```

In the code, `a` as well as `b` have been passed as arguments to the function. `a` and `b` are local variables that can be accessed only within the function, as they exist in the function. However, `z` is declared outside the function and hence is a global variable.

We can see the output of the executed code in the following screenshot:

Variables in JavaScript

After subtraction, the difference is 3

Functions will be explained under the heading *Functions* in this book, so let's stick to variables as of now.

To understand the scope and concept of variables better, let's have a look at the following code:

```
<html>
<head>
<title> Scope of variables </title>
</head>
<body>
<script type="text/javascript">
  var a = "Packt publishes Books on Technology";
  var b = "Welcome to Packt";
  function Packt()
  {
    var a = "Packt Publishing has an online book library";
    alert(a);
    alert(b);
  }
  Packt();
  alert(a);
</script>
</body>
</html>
```

On execution, we see three alerts in the following order :

- ▶ Packt Publishing has an online book library
- ▶ Welcome to Packt
- ▶ Packt publishes Books on Technology

From the output, it is quite evident that the function can access the local variable (hence the first alert) as well as the global variable (hence the second alert). However, anything outside a specific function cannot access the variables inside that function (hence the third alert). That clears the concept of global and local variables.

Operators

The three types of commonly used operators in JavaScript are as follows:

- ▶ Arithmetic operators
- ▶ Logical operators
- ▶ Comparison operators

Arithmetic operators

The following table displays the various kinds of Arithmetic operators:

Arithmetic Operators	
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modular	%
Increment	++
Decrement	--

The operators mentioned in the table are used for Arithmetic operations.

The **Addition**, **Subtraction**, **Multiplication**, and **Division** operators are self explanatory. The **Modular** operator (%) is used for the remnant left after division (we call this the modulus). Increment operators are used to increase the value by one whereas the decrement operator is used to decrease the value by one.

For example, $x = x + 1$ can also be written as $x++$ whereas $y = y - 1$ can be written as $y--$.

Logical operators

The following table displays the different types of logical operators:

Logical Operators	
AND	& &
OR	
NOT	!

The **AND** operator is used when all the conditions must be satisfied. For example, `x>5 && y=5` means that `x` should be greater than 5 and `y` must be equal to 5 for the condition to be true.

The **OR** operator is used when one of the conditions must be true. For example, `x=5 || y<3` means that the condition is true when `x` is equal to 5 or `y` is less than 3.

The **NOT** operator is used when the antithesis is true. For example, `!(x>5)` is true when `x` is equal to or less than 5.

Conditions and Loops

The main purpose of JavaScript is to add dynamic functionality. In order to execute a lot of actions and iterations, we have the concept of conditions and loops in JavaScript. The `if-else` statement is primarily used for various conditions. The structure of the `If-else` statement is as follows:

```
if (condition)
{
    JavaScript statements to be executed
}
else
{
    JavaScript statements to be executed
}
```

For example, consider the following code snippet:

```
If (i>5)
{
    x=50;
}
else
{
    x=100;
}
```

The interpretation of this code is simple. If the value of `i` is greater than 5, the value of `x` will be 50. If the value of `i` is not greater than 5, then the value of `x` will be 100.

At times, we know the number of iterations to be performed. Suppose we need to execute the code and get the output, say 10 times. At that time, we use loops to perform multiple iterations as we know the number of times the loop will be repeated. We will see an example of the `for` loop in this chapter.

The structure of the `for` loop is as follows:

```
for (initialization;condition;increment)
{
    JavaScript statements
}
```

Let us have a look at a code snippet:

```
<!DOCTYPE html>
<html>
<body>
<script type="text/javascript">
    for (i=0;i<5;i++)
    {
        document.write(i+ "<br>");
    }
</script>
<p>Hi</p>
<p>This is an example of the execution of the for loop</p>
</body>
</html>
```

The output of the code on execution would be as follows:

```
0
1
2
3
4

Hi

This is an example of the execution of the for loop
```

Functions

Functions are an integral part of JavaScript. Let's understand the concept of functions. We know that the imperative purpose of JavaScript is to add functionality to the web page. Say, we want to add certain functionality in JavaScript. Suppose we need to implement it on different pages of the website or different places on the same web page. Instead of adding functionality each time, we can use the concept of functions where we can declare the function once. From there on, we can use it multiple times on multiple web pages or at different places on the website.

There are many inbuilt functions in JavaScript. We can also craft our own functions in JavaScript and customize them according to our needs. Let's look at the following code to understand how the functions work:

```
<!DOCTYPE html>
<html>
<head>
<title> Java Script in a nutshell </title>
<h1> Functions in JavaScript </h1>
<p> Click on "Click Here" to execute the code in the function </p>
<script type="text/javascript">
    function PacktPub()
    {
        var x = prompt("Hey", "");
        alert("Welcome to Packt, " + x);
    }
</script>
</head>
<body>
<a href="javascript:PacktPub()">Click Here</a>
</body>
</html>
```

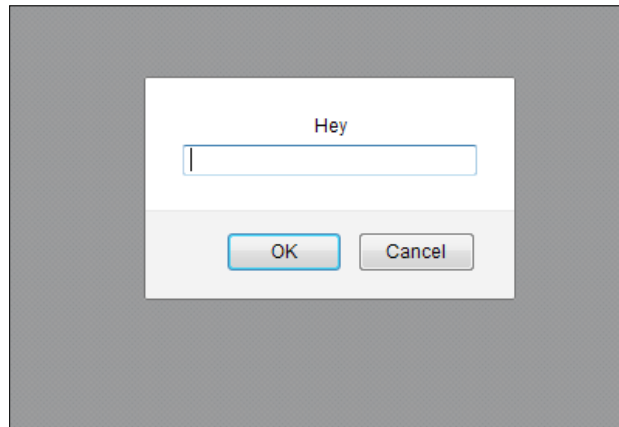
The execution of the code will give us the following output:

Functions in JavaScript

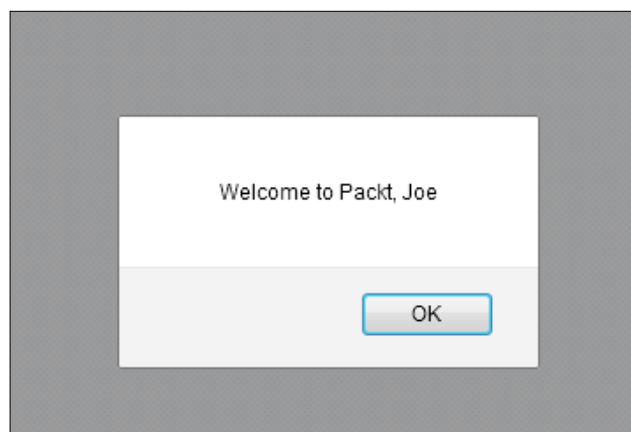
Click on "Click Here" to execute the code in the function

[Click Here](#)

On clicking the **Click Here** link, we can see the following output:





We get a prompt box that requests for an input. Say we enter the name **Joe** and click on **OK**. We will get the following alert box as defined in the code.



Let's now understand how the code works. We have used two inbuilt functions in the code:

- ▶ `alert`
- ▶ `prompt`

`alert` and `prompt` are inbuilt functions in JavaScript.

 JavaScript is case-sensitive; hence, we must use lowercase for `alert` and `prompt`, otherwise the code will not work. 

The `alert` and `prompt` functions are commonly used in JavaScript. An alert box is used when we need to inform the user about some development. It is also used when we want to give the user some information prior to clicking on a link or surfing the website. Similarly, a prompt box is used when we want an input from the user before he accesses the page.

If you observe the code, it is quite evident that the `PacktPub()` function was declared prior to calling it. In the `PacktPub()` function, we have used the `prompt` and `alert` functions. The `prompt` command on execution will prompt the user for an input. The double quotes (" ") after `Hey` are for the default string that has to be entered as the input. In the code, `x` is a variable that is assigned a `prompt` function. Again, `x` is used in the `alert` command in the very next line.

Once we enter the name `Joe`, we get an alert: `Welcome to Packt, Joe`.

The concept of functions must be clear by now. Functions are used extensively in JavaScript and are widely used to return a value in real-world programming. Hence, functions are an imperative part of JavaScript.

Objects and their role in JavaScript

JavaScript is an object-oriented language. We have inbuilt objects such as `Math`, `Date`, and `Document`, to mention a few. However, we can also create our own objects and customize them to suit our needs.

An object has properties and methods to define it. These are attributes that give meaning to the object-oriented concept in JavaScript. A **property** is an attribute that defines values for the object. Methods define the action that can be implemented on the objects.

Let's have a look at the following code to understand how properties define values for an object:

```
<!DOCTYPE html>
<html>
<body>
<script>
    var animal=new Object();
    animal.species="Dog";
    animal.nickname="Monster";
    animal.age=10;
    animal.eyecolor="brown";
    document.write(animal.nickname + " is a dog that has " +
    animal.eyecolor + " eyes");
</script>
</body>
</html>
```

The output of the code would be as follows:

Monster is a dog that has brown eyes

Thus, we have defined properties for a dog called Monster. If you observe the code and the output, we can see that we have connected the properties using string values connected by an operator known as the concatenation operator. The concatenation operator is used to connect several pieces of text together. One point to remember is that the empty space in the string is taken into consideration. Observe how we have a space preceding and following the " is a dog that has " string and how it reflects in the output. Coming back to objects, we have defined the properties for the dog by specifying its name, age, and eye color.

In JavaScript, all the user-defined objects and the inbuilt objects belong to an object called `Object`. Let's understand the concept of the `new` operator and the constructor function. In the earlier code, there is a line of code `var animal=new Object();`. Hence, it is evident that the `new` operator is used to create an instance of the object. The constructor used in this case is `Object()`. The `new` operator is always followed by a constructor method. That explains the code structure to create a new object.

Instead of the inbuilt functions, we can also use user-defined functions. Let's have a look at the following code to understand how it works:

```
<html>
<head>
<script type="text/javascript">
  function rate(cost){
    this.sellingPrice = cost;
  }
  function prose(name, company){
    this.name= name;
    this.company = company;
    this.rate = rate;
  }
</script>
</head>
<body>
<script type="text/javascript">
  var y = new prose("JavaScript", "Packt Publishing");
  y.rate(50);
  document.write("The name of the book is " + y.name+ "<br>");
```

```
document.write("The name of the publishing company is : " +  
y.company + "<br>");  
document.write("The cost of the book is $" + y.sellingPrice +  
"<br>");  
</script>  
</body>  
</html>
```

The output of the code would be as follows:

The name of the book is JavaScript
The name of the publishing company is : Packt Publishing
The cost of the book is \$50

We need to understand that a function and method are very similar. The only difference is that a **function** is a standalone bunch of statements whereas a **method** belongs to an object and can be referenced by the `this` keyword. In the code, we have defined the `rate` function that will act as the method; then we assign the `rate` method as a property in the function prose. Hence, we have seen the concept of properties and methods in JavaScript objects.

As we mentioned, JavaScript is an object-oriented language. As with any object-oriented language, the OOP concept applies to JavaScript as well. **Encapsulation**, **Inheritance**, **Abstraction**, and **Polymorphism** are features of OOP that can be applied to JavaScript.

Arrays

In many programming languages such as C++, Java, and PHP to mention a few, we use the concept of arrays. Similarly, we use the concept of arrays in JavaScript too. Objects in JavaScript store and allow manipulation of data. So far, we have seen variables that are restricted to a single piece of data. However, in an array, we can store multiple items and retrieve them with ease.

Arrays can be created in different ways. Let's have a look at the following code snippet to understand it better:

```
var animals = new Array();  
animals[0] = "cat";  
animals[1] = "dog";  
animals[2] = "mouse";
```

We can also create an array by entering the items directly within the `new Array` constructor. The following code snippet will show how it works:

```
var animals = new Array("cat", "dog", "mouse");
```

We can also define an array without using the `new Array` constructor.

```
var animals = ["cat", "dog", "mouse"];
```

The index position of an array starts with 0. The index feature helps us in accessing the data stored in the array. Let's say we define the array as follows:

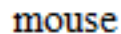
```
var animals = ["cat", "dog", "mouse"];
```

If we want to retrieve the third item in the array, we need to specify its index position.

Let's have a look at the following code to understand how it works:

```
<html>
<head>
<title> Index in Arrays </title>
</head>
<body>
<script type= "text/javascript">
var animals = ["cat", "dog", "mouse"];
var x = animals[2];
document.write(x);
</script>
</body>
</html>
```

The output of the following code would be as follows:



An array has inbuilt properties and methods. For example, `length` is a property of arrays. Let's have a look at the following code to see how it works:

```
<html>
<head>
<title> Index in Arrays </title>
</head>
<body>
<script type= "text/javascript">
var animals = ["cat", "dog", "mouse"];
y=animals.length
```

```
document.write(y);  
</script>  
</body>  
</html>
```

The output of the code would be 3.

In brief, let's look at the various properties and methods of arrays:

- ▶ **push:** To add an item at the end of the array
- ▶ **pop:** To remove an item at the end of the array
- ▶ **reverse:** To reverse the order of elements in the array
- ▶ **toString:** To convert an array to a string
- ▶ **sort:** To sort items alphabetically or numerically in an ascending or descending order

An overview on jQuery – the need of the hour

jQuery, a multi-browser JavaScript library, was invented by John Resig. It is an open source library. Modern complex websites that have a lot of animation and navigation can be developed with ease using jQuery. **jQuery mobile (JQM)** is also an offshoot of jQuery. With the advent of smartphones and tablets, responsive web design is the trend today. Responsive web design can be achieved with the use of jQuery.

Adding jQuery to the web page

jQuery can be added to the website in the following ways:

- ▶ Including jQuery locally by downloading a copy of the library
- ▶ Including a hosted copy of jQuery

Let's have a look at both of the procedures in the next sections.

Including jQuery locally by downloading a copy of the library

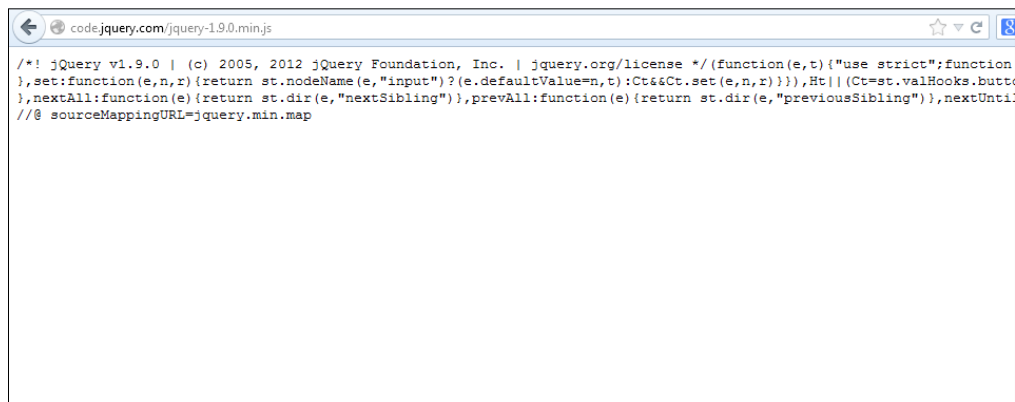
When we visit the jQuery website at <http://jquery.com/> to download the copy of the library, we can view two different versions. The first version in the download section is the Minified Production jQuery version, whereas the second version is the Uncompressed Development version. There is a difference between the two.

The Uncompressed Development version is readable and has lots of whitespaces and comments to understand the code better. Hence, it is recommended you use the Uncompressed Development version when you are learning jQuery or are in the development stage.

The Minified Production version is precise and does not have the whitespaces as in the Uncompressed Development version. Moreover, the variables are given short names and the comments removed along with other compression tricks to reduce the size of the file as much as possible. Hence, during deployment, it is advisable to use the Minified version, as a shorter file has to be downloaded on loading the page. Let's see how we add the jQuery file locally.

When we go to the website, we can view the Minified Production and Uncompressed Development versions. Click on the version of your choice and it will be loaded in the browser as shown in the following figure.

In our case, we will download the Minified version:



Copy the contents of the file and save it in a text file giving it an extension .js. In our case, we will save it as latestjquery.js. Make sure you are saving the file in the same directory as your project so that we can call it directly. The following code snippet displays the way we include jQuery in our website:

```
<!DOCTYPE html>
<html>
<head>
<title>Including jQuery</title>
<script type="text/javascript" src="latestjquery.js"></script>
<script type="text/javascript"
src="ourjavascriptpage.js"></script>
</head>
<body>
```

```
<p>jQuery: an amazing JavaScript library</p>
</body>
</html>
```

In the preceding code, we have invoked `latestjquery.js` (our jQuery file) prior to any JavaScript code for example, (`ourjavascriptpage.js`). Hence, it is quite evident that we need to include jQuery first as our code needs to find jQuery fully loaded before we use it. It is good practice to include the jQuery file in our `<head>` section as it has to be loaded completely. Let's now have a look at the second way of including jQuery to our website.

Including a hosted copy of jQuery

We can use the **Content Delivery Network (CDN)** of Google or Microsoft to host the jQuery file. Let's have a look at the following code snippet to see how it works.

If we go to the Google CDN, we can see the following code snippet that has to be included in order to use jQuery:

```
<script
src="//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js">
</script>
```

If we go to the Microsoft CDN, we can see the following code snippet that has to be included in order to use jQuery:

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-
1.9.0.js"></script>
```

We also have the minified version that can be used:

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-
1.9.0.min.js"></script>
```

Remember that we need to include this snippet before our JavaScript file as jQuery has to be loaded completely if we need to use it.

Using the document ready feature

Before we understand the document ready feature, let's look at the following code:

```
<html>
<head>
<title>Understanding the document ready function</title>
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-
1.9.0.js"></script>
<script type="text/javascript">
```

```
$( "p" ).click(function() {  
    alert("Welcome to Packt Publishing");  
});  
</script>  
</head>  
<body>  
<p>Click here to understand the document ready feature</p>  
</body>  
</html>
```

On execution, we can see the following output:

Click here to understand the document ready feature

According to the code, when we click on the text, an alert dialog box is displayed that reads Welcome to Packt Publishing.

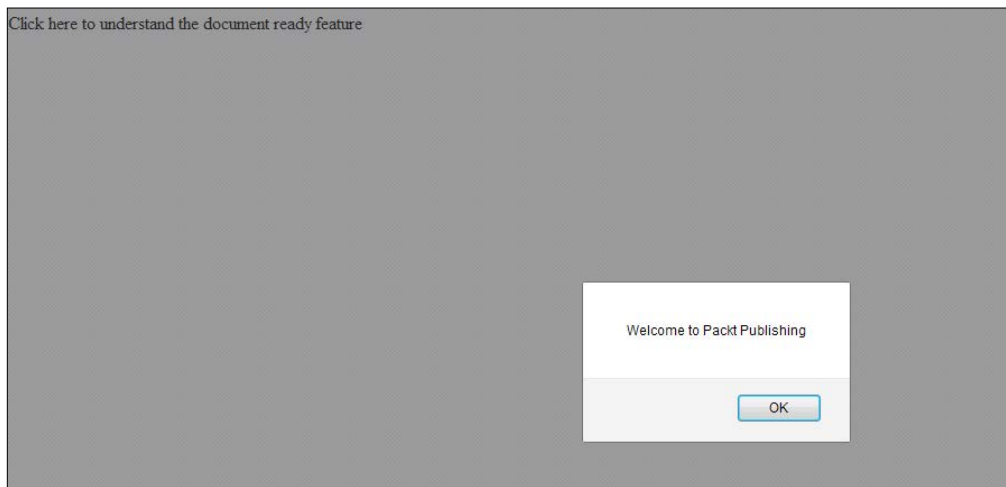
However, we do not see any alert box. This is because the events must be fired, or should perform the desired action, before the page loads. If you observe the code, the alert box is not displayed as the method is not called before the `click` event takes place. By using the document ready feature, we can solve this issue. Let's tweak the code a bit and include the document ready feature.

```
<html>  
<head>  
<title>Understanding the document ready function</title>  
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.9.0.js">  
</script>  
<script type="text/javascript">  
$(document).ready(function() {  
    $( "p" ).click(function() {  
        alert("Welcome to Packt Publishing");  
    });  
});  
</script>  
</head>  
<body>  
<p>Click here to understand the document ready feature</p>  
</body>  
</html>
```

The code on execution will display the following output:

Click here to understand the document ready feature

However, the difference here is that on clicking the text in the output, the following alert box gets displayed:



Hence, it is a good practice to include the document ready feature in jQuery. We have to use the following syntax to implement the document ready feature:

```
$(document).ready(function()
```

If we observe the code snippet, the function is the callback function that has to be executed once the document gets ready for implementing jQuery. We can also observe a \$ sign. The \$ sign is the synonym for the `jQuery()` function. Instead of writing `jQuery` before the tag name, tag id, or class, we can use `$`. It simplifies and shows that it is a jQuery code snippet. Using the appropriate syntax makes up for cleaner and shorter code; this is of prime importance when the page loads.

Accessing DOM elements using jQuery

Similar to CSS, we can access elements in the **Document Object Model (DOM)** using jQuery. In JavaScript, we use the `getElementById` function to access elements in the DOM by using their ID. However, we can do that in jQuery in a much shorter code.

Using the ID

Let's have a look at the following code snippet to see how we access the elements by their IDs:

```
$(document).ready(function()
{
    var x = $("#packt");
});
```

For an element with an ID `packt`, we can refer to it using the `#` operator similar to the way it is done in CSS. We need to remember that the ID has to be preceded by a `#` and should be enclosed in double quotes.

Using the class

Let's have a look at the following code snippet to see how we access the elements by their class:

```
$(document).ready(function()
{
    var y = $(".pubman");
});
```

For an element defined with a class `pubman`, we can refer it using the dot (`.`) operator. The class has to be preceded by a dot (`.`) operator similar to the way it is done in CSS. However, the class has to be enclosed in double quotes.

Using the tag

Let's have a look at the following code snippet to see how we access the elements by their tag name:

```
$(document).ready(function()
{
    var links = $("a");
});
```

The elements can be accessed by their tag name in jQuery by referring to the tag enclosed in double quotes.

Anonymous functions

Let's have a look at the way we use anonymous functions in jQuery.

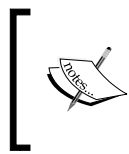
The following code will give us an idea about its functionality:

```
$(document).ready(function()
{
```

```
$('#packt').click(function()
{
    $('#div').remove();
});
});
```

Let's understand the meaning of the code.

The document ready feature has already been explained. Then we query the DOM for the element with the ID `packt` and we assign a `click` event to it. Inside the function, we have included the `remove` method, which will remove all `<div>` elements.



For the various methods used in jQuery, we can have a look at books such as *jQuery for Designers: Beginner's Guide*, which are on the Packt website at www.packtpub.com. Alternatively, we can refer to the Packt online library at <http://packtlib.packtpub.com/>.

jQuery is widely used for animation. We will now have a look at the following code where we hide and show elements on the web page:

```
<html>
<head>
<title>jQuery Animation</title>

<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.9.0.js">
</script>
<script type="text/javascript">

    $(document).ready(function() {
        $("#packt").click(function () {
            $(".pubman").show( 2000);
        });

        $("#packtpub").click(function () {
            $(".pubman").hide( 3500 );
        });

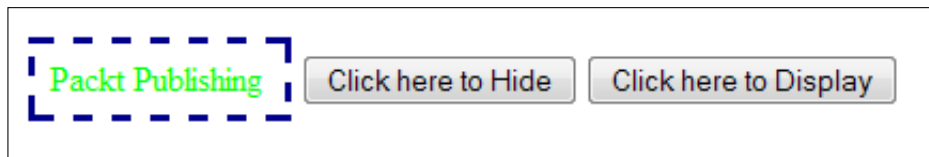
    });
</script>
<style>
.pubman{ margin:5px;padding:7px;
border:3px dashed navy;
color:lime;
```

```
        width:110px;
        height:40px;
        display:inline;
    }
</style>
</head>
<body>
<br>
<div class="pubman">
Packt Publishing
</div>

<input id="packtpub" type="button" value="Click here to Hide" />
<input id="packt" type="button" value="Click here to Display" />

</body>
</html>
```

Let's have a look at the output of the code on execution:



When we click on **Click here to Hide**, the box containing the text **Packt Publishing** vanishes. When we click on **Click here to Display**, the box appears again. We have used the `show()` method as well as the `hide()` method in the jQuery code. We have also specified the time in milliseconds for the transition to take place.

Hence, we can see how easy it is to create animations using jQuery. There are various other methods such as `toggle` and `fade`, to mention a few, but we cannot cover everything that jQuery has to offer in this book.

jQuery is a vast subject and the book will only look at a certain aspect of it. Packt has a lot of books on jQuery such as *jQuery for Designers: Beginner's Guide*, Packt Publishing, and you can visit our web page at <http://www.packtpub.com/books?keys=jquery> to see the various books on jQuery customized for your needs.

Coding gets better with practice. We would really appreciate if you copy the code mentioned in this book on an editor such as Notepad or Notepad++ and practice it. When you implement the things you have learned in this book in real life, you will realize how interesting it is to work on JavaScript practically. We have covered objects and arrays in this book. There are different kinds of objects such as the `Math` object and `Date` object, to mention a few. However, everything cannot be covered here.

If the right book for you isn't listed here, you can visit our online library PacktLib at <http://packtlib.packtpub.com/> for our full list of JavaScript titles as well as a wide range of other technologies. Alternatively, you can check out our responsive browser app at <https://app.packtpub.com/> using your smartphone or tablet.

Feel free to let us know about the kind of titles that you'd like to see on JavaScript. You can get in touch via contact@packtpub.com.

Your input is very important to us; it helps us produce better products that help the community more. If you have any feedback on the books, or are struggling with something we haven't covered, surely let us know. You can e-mail us at customercare@packtpub.com for the same.

To know more about our future releases, our full catalogue, and daily deals, please visit <http://www.packtpub.com/>.