






## Reading Sample

*In these sample chapters you'll first learn about the ways in which you can embed JavaScript in a web page and generate simple output, and then see how you can use the jQuery library to simplify different JavaScript programming tasks.*

-  **"Getting Started"**
-  **"Simplifying Tasks with jQuery"**
-  **Contents**
-  **Index**
-  **The Author**

Philip Ackermann

### JavaScript: The Comprehensive Guide

982 pages, 2022, \$59.95  
ISBN 978-1-4932-2286-5

 [www.rheinwerk-computing.com/5554](https://www.rheinwerk-computing.com/5554)

## Chapter 2

# Getting Started

*JavaScript is still mainly used for creating dynamic web pages—within a browser. Before we take a closer look at other application areas in later chapters, this chapter will show you the ways in which you can embed JavaScript in a web page and generate simple output. This chapter thus is the basis for the following chapters.*

Before we go into further detail about the JavaScript language itself, you should first know how JavaScript relates to HTML and CSS within a web page, how to embed JavaScript in a web page, and how to generate output.

### 2.1 Introduction to JavaScript and Web Development

The most important three languages for creating web frontends are certainly HTML, CSS, and JavaScript. Each of these languages serves its own purpose.

#### 2.1.1 The Relationship among HTML, CSS, and JavaScript

In HTML, you use *HTML elements* to specify the *structure* of a web page and the *meaning* (*semantics*) of individual components on a web page. For example, they describe which area on the web page is the main content and which area is used for navigation, and they define components such as forms, lists, buttons, input fields, or tables, as shown in Figure 2.1.

Artist	Album	Release Date	Genre
Monster Magnet	Powertrip	1998	Spacerock
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Ben Harper	The Will to Live	1997	Singer/Songwriter
Tool	Lateralus	2001	Progrock
Beastie Boys	Ill Communication	1994	Hip Hop

Figure 2.1 HTML Is Used to Define the Structure of a Web Page

CSS, on the other hand, uses special *CSS rules* to determine how the individual components that you have previously defined in HTML should be displayed; this is used to define the *design* and *layout* of a web page. For example, you can define text color, text size, borders, background colors, color gradients, and so on. Figure 2.2 shows how CSS was used to adjust the font and font size of the table headings and table cells, add borders between table columns and table rows, and alternate the background color of the table rows. The whole thing looks a lot more appealing than the variant without CSS.

Artist	Album	Release Date	Genre
Monster Magnet	Powertrip	1998	Spacerock
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Ben Harper	The Will to Live	1997	Singer/Songwriter
Tool	Lateralus	2001	Progrock
Beastie Boys	Ill Communication	1994	Hip Hop

Figure 2.2 With CSS, You Define the Layout and Appearance of Individual Elements of the Web Page

Last but not least, JavaScript is used to add *dynamic behavior* to the web page (or to the components on a web page) or to provide more interactivity on the web page. Examples of this are sorting and filtering the table data, as already mentioned in Chapter 1 (see Figure 2.3 and Figure 2.4). So while CSS takes care of the design of a web page, JavaScript can be used to improve the user experience and interactivity of a web page.

Q

Search artist

Artist ▾	Album	Release Date	Genre
Beastie Boys	Ill Communication	1994	Hip Hop
Ben Harper	The Will to Live	1997	Singer/Songwriter
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Monster Magnet	Powertrip	1998	Spacerock
Tool	Lateralus	2001	Progrock

Figure 2.3 Sort Option to Make a Web Page More User-Friendly and Interactive with JavaScript

Q

Be

Artist ▾	Album	Release Date	Genre
Beastie Boys	Ill Communication	1994	Hip Hop
Ben Harper	The Will to Live	1997	Singer/Songwriter

Figure 2.4 Filter Option to Make a Web Page More User-Friendly and Interactive with JavaScript

Thus, in the vast majority of cases, a web page consists of a combination of HTML, CSS, and JavaScript code (see Figure 2.5). Note that though we just said that JavaScript takes care of the behavior of a web page, you can create functional web pages entirely without JavaScript. In principle, you can also create web pages without CSS; it is possible. In that case, only the HTML is evaluated by the browser. That means, however, that the web page is less fancy (without CSS) and less interactive and user-friendly (without JavaScript), as shown previously in Figure 2.1.

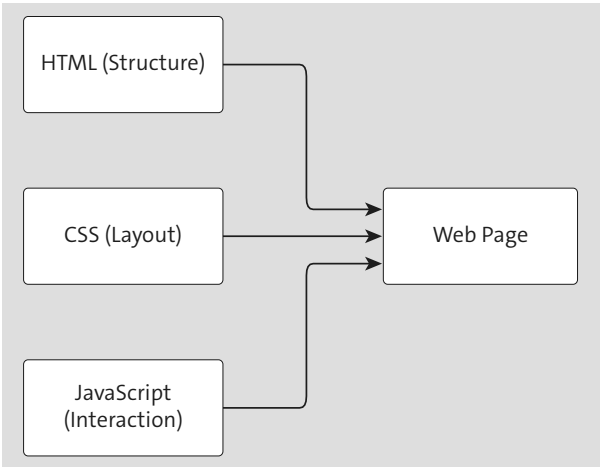


Figure 2.5 Usually, a Combination of HTML, CSS, and JavaScript Is Used within a Web Page

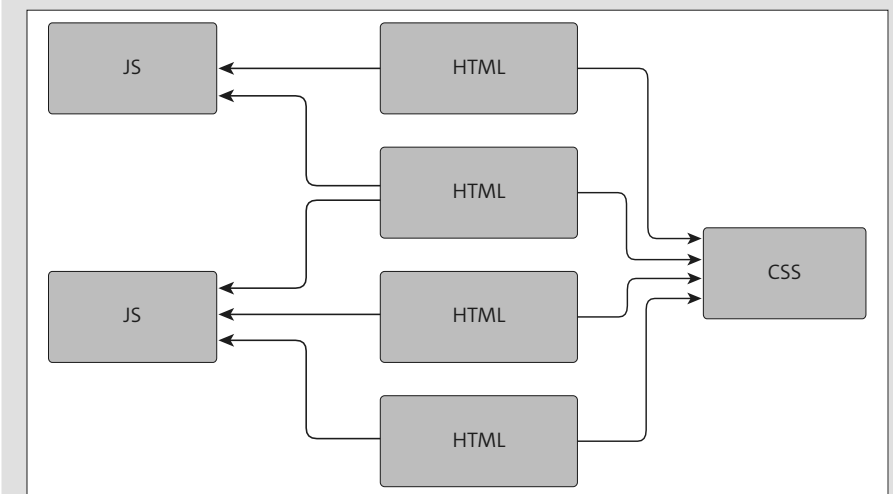
**Note**  
HTML is used for the structure of a web page, CSS for layout and design, and JavaScript for behavior and interactivity.

Definition

Web and software developers also refer to three layers in this context: HTML provides the *content layer*, CSS the *presentation layer*, and JavaScript the *behavioral layer*.

Separating the Code for the Individual Layers

It is considered good development style not to mix the individual layers—that is, to keep HTML, CSS, and JavaScript code independent of each other and in separate files. This makes it easier to keep track of a web project and ultimately ensures that you can develop more effectively. In addition, this method enables you to include the same CSS and JavaScript files in various HTML files (see Figure 2.6) and thus to reuse the same CSS rules or JavaScript source code in several HTML files.



**Figure 2.6** If You Write CSS and JavaScript Code into Separate Files rather than Directly into the HTML Code, It Is Easier to Reuse

A good approach to developing a website is to think about its structure first: What are the different areas of the web page? What are the headings? Is there any data presented in tabular form? What are the navigation options? Which information is included in the footer area and which in the header area of the page? Only HTML is used for this purpose. The website won't look nice or be very interactive, but that isn't the point of this first step, in which we do not want to be distracted from the essential element: the website content.

Building on this structural foundation, you then implement the design using CSS and the behavior of the web page using JavaScript. In principle, these two steps can also be carried out in parallel by different people. For example, a web designer may take care of

the design with CSS, while a web developer programs the functionality in JavaScript (in practice, the web designer and web developer are often one and the same person, but especially in large projects with numerous websites, a distribution of responsibilities is not uncommon).

Phases of Website Development

When developing professional websites, there are several stages preceding the development step. Before development even begins, prototypes are designed in concept and design phases (either digitally or quite classically with pen and paper). The step-by-step approach just described (first HTML, then CSS, then JavaScript) thus only refers to development.

HTML Markup Language and CSS Style Language

By the way, HTML and CSS are not programming languages! HTML is a *markup language* and CSS is a *style language*; only JavaScript of the languages we're discussing here is a *programming language*. Strictly speaking, statements like "This can be programmed with HTML" are therefore not correct. You'd instead have to say something like "This can be realized with HTML."

Definition

The process of presenting a web page in the browser is called *rendering*. A common phrase among developers is "The browser renders a web page." This involves evaluating HTML, CSS, and JavaScript code, creating an appropriate model of the web page (which we'll talk about in Chapter 5), and "drawing" the web page into the browser window. In detail, this process is quite complex, and if you're interested in this topic, you might want to read the blog post at [www.html5rocks.com/en/tutorials/internals/howbrowserswork](http://www.html5rocks.com/en/tutorials/internals/howbrowserswork).

2.1.2 The Right Tool for Development

In principle, a simple text editor would be sufficient for creating JavaScript files (and for simple code examples this is perfectly fine), but sooner or later you should acquire a good editor that supports you when writing JavaScript and that is specifically designed for developing JavaScript programs (if you don't already have one installed on your computer anyway). Such an editor supports you, for example, by highlighting the source text in color, relieving you of writing recurring source text modules, recognizing errors in the source text, and much more.



Editors

There are a number of really good editors that can be used effectively. For example, Sublime Text ([www.sublimetext.com](http://www.sublimetext.com); see Figure 2.7) and Atom (<https://atom.io>; see Figure 2.8), both available for Windows, macOS, and Linux, are popular editors in the developer community. While the former currently costs \$99 (as of June 2021), the Atom editor is free of charge. In detail, both editors have their own features and strengths, but they are still quite similar. Try them out to see which one suits you more.

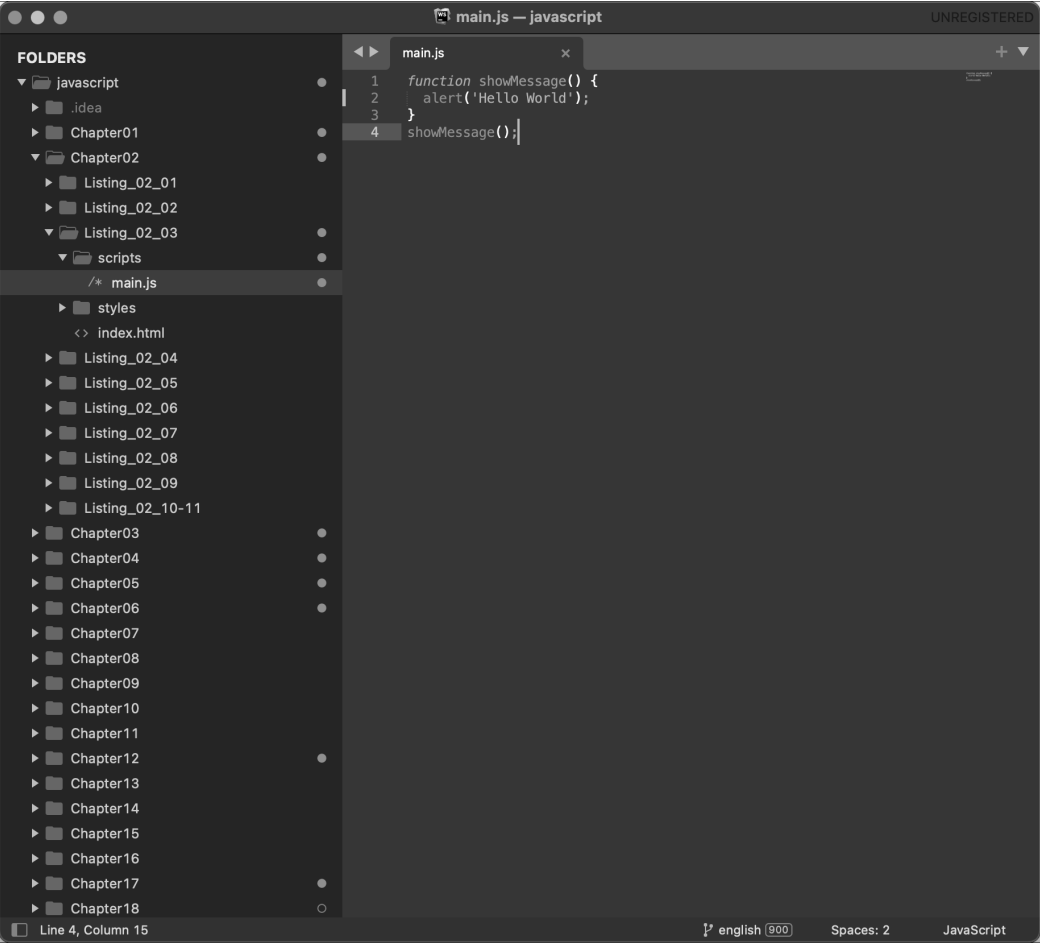


Figure 2.7 Sublime Text Editor

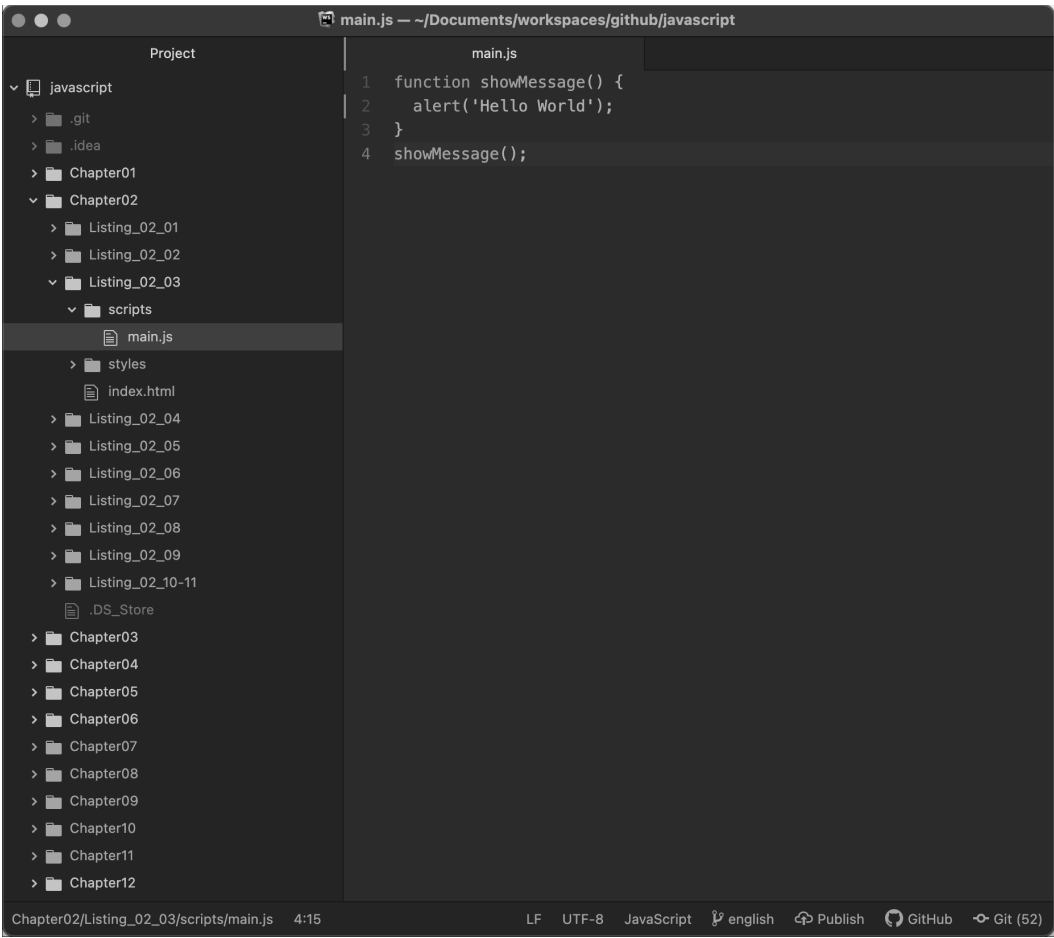


Figure 2.8 Atom Editor

Development Environments

Software developers switching from languages like Java or C++ to JavaScript are in most cases used to integrated development environments (IDEs), as known from their previous programming languages. In a way, you can think of a development environment as a very powerful editor that provides various additional features compared to a "normal" editor, such as synchronization with a source control system, running automatic builds, or integrating test frameworks. (If you're just shaking your head uncomprehendingly now and wondering what's behind all these terms, wait until Chapter 21, in which we'll go into more detail about these advanced topics of software development with JavaScript.)

WebStorm by IntelliJ ([www.jetbrains.com/webstorm/](http://www.jetbrains.com/webstorm/); see Figure 2.9) is one example of a very popular and also very good development environment. A single license for WebStorm currently costs USD 129 (for personal use, there is another version that currently costs USD 59). However, if you want to test the program first, you can download a 30-day trial version from the WebStorm homepage. WebStorm is available for Windows and for macOS and Linux.

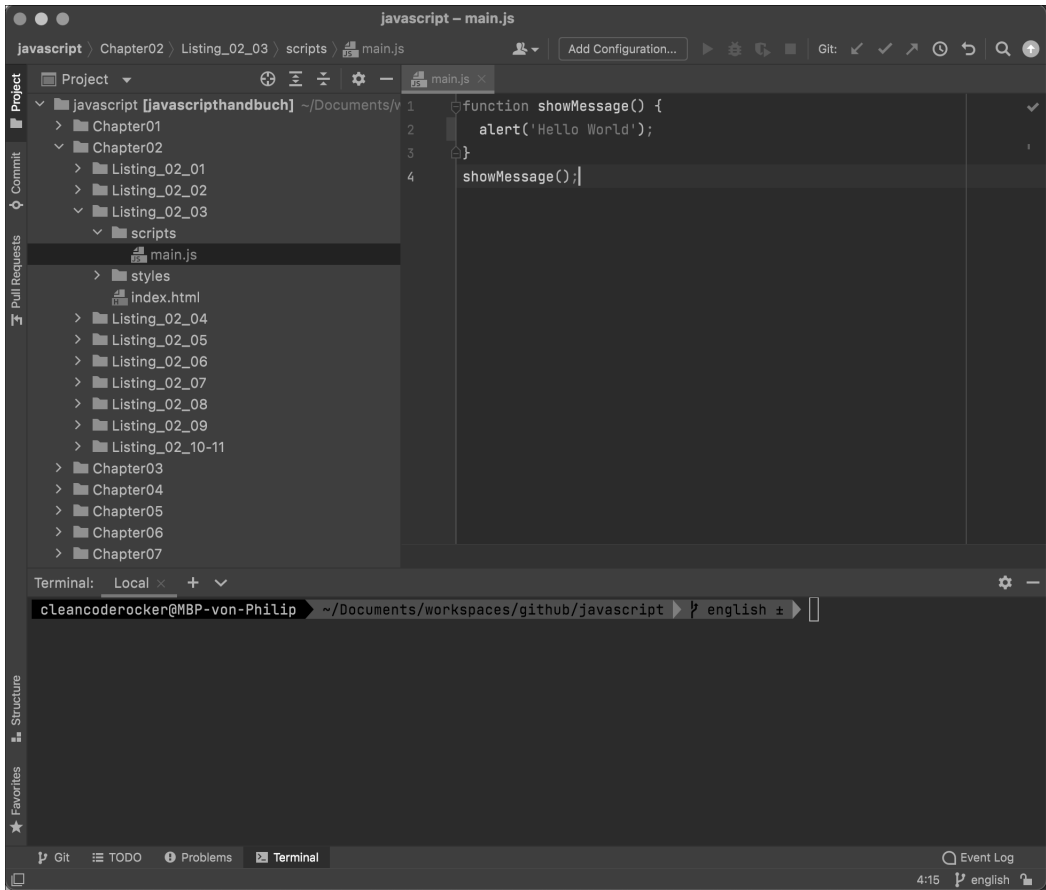


Figure 2.9 WebStorm IDE

Meanwhile a personal favorite among the development environments is Visual Studio Code by Microsoft (<https://code.visualstudio.com>; see Figure 2.10). It is available for download free of charge, can be flexibly extended via plug-ins, and its perceived performance is significantly better than that of WebStorm, for example.

A brief overview of the editors and development environments we’ve discussed is shown in Table 2.1.

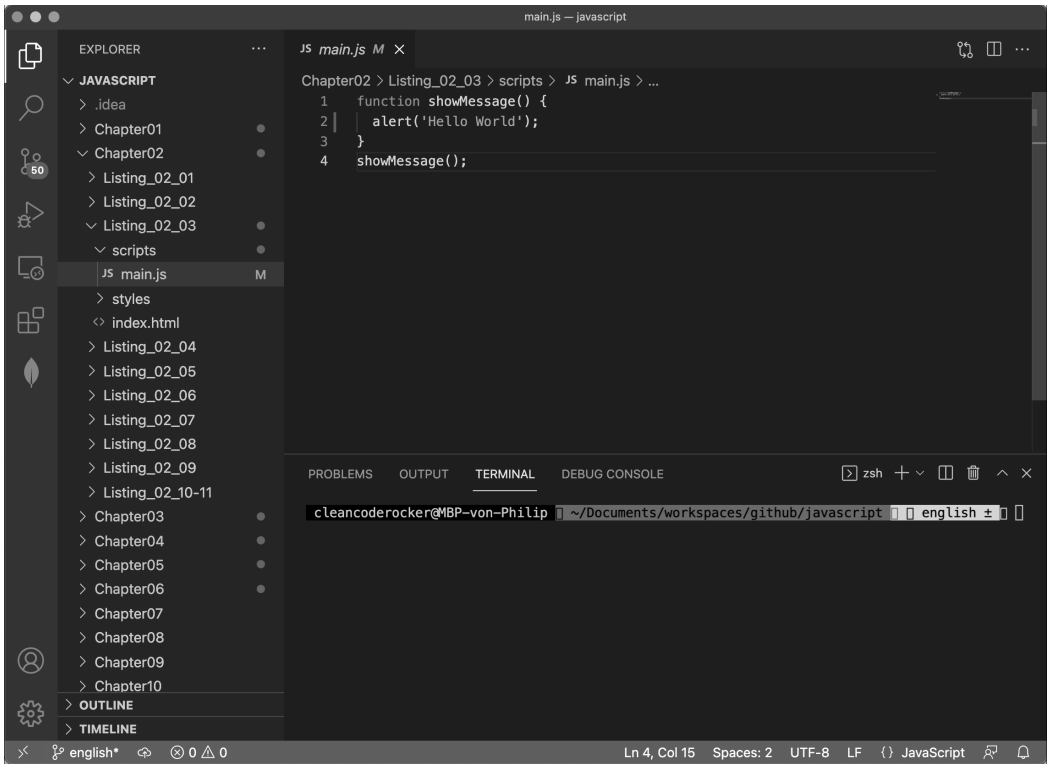


Figure 2.10 Microsoft Visual Studio Code

Name	Price	macOS	Linux	Windows	Editor/Development Environment
Sublime Text	USD 99	Yes	Yes	Yes	Editor
Atom	Free of charge	Yes	Yes	Yes	Editor
Microsoft Visual Studio Code	Free of charge	Yes	Yes	Yes	Development environment
WebStorm	USD 129/ USD 59	Yes	Yes	Yes	Development environment

Table 2.1 Recommended Editors and Development Environments for JavaScript Development

Tip

For the beginning—for example, for trying out the code examples in this book—we recommend that you use one of the editors mentioned in this section and not a

development environment (yet). The latter have the disadvantage that they are partly overloaded with menus and functionalities, so you have to deal not only with learning JavaScript but also with learning the development environment. Let's spare you that at least for the moment.

In addition, development environments only make sense when exceeding a certain project size. For smaller projects and the examples in this book, an editor is always enough (even though we will also cover complex topics). Plus, the editors are usually faster than the development environments in terms of execution speed.

## 2.2 Integrating JavaScript into a Web Page

Because we assume that you already know how to create an HTML file and how to embed a CSS file, and that you are "only" here to learn JavaScript, we don't want to waste any more time with details about HTML and CSS but will get started with JavaScript straight away. Don't worry: embedding and executing a JavaScript file is anything but difficult.

### Learn HTML and CSS

If you have not worked with HTML or CSS a very good introductory book on this topic is *HTML and CSS: Design and Build Websites* by Jon Duckett (2011, John Wiley & Sons).

Per tradition (like almost every book on programming languages), we will start with a very simple *Hello World* example, which only produces the output `Hello World`. This is not very exciting yet, but right now the point is to show you how to embed a JavaScript file in an HTML file in the first place and how to execute the source code contained in the JavaScript file. We will take care of more complex things later.

### 2.2.1 Preparing a Suitable Folder Structure

For getting started and working through the following examples, we recommend that you use the directory structure shown in Figure 2.11 for every example. The HTML file is at the top level because this is the entry point for the browser and thus the file you will invoke in the browser right away.

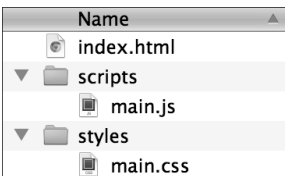


Figure 2.11 Example Folder Structure

However, it is a good idea to create different folders for the CSS and JavaScript files. The names *styles* (for CSS files) and *scripts* (for JavaScript files) are quite common. Especially if you are dealing with a lot of different JavaScript and CSS files during development, this separation (or an arrangement with subfolders in general) makes it easier to keep track of your project.

### Starting Point of a JavaScript Application

Most of the examples in this book also follow the layout shown in Figure 2.11 as we will only run the JavaScript code in the browser at the beginning, using the *index.html* file as a kind of entry point to the program.

Later, in Chapter 17, you'll learn how you can also run JavaScript independent of a browser and thus independent of a corresponding HTML file. In this case, you don't need any HTML—and therefore no CSS files either.

### Running JavaScript in the Browser

While you can execute JavaScript within a browser without creating an HTML file to embed the corresponding script (via special developer tools provided by browsers; Section 2.3.2), for now we don't want to use this feature.

### 2.2.2 Creating a JavaScript File

As mentioned earlier, it's better to save JavaScript code in a separate file (or in several separate files) that can then be embedded in the HTML code. So the first thing you need is a JavaScript file. Simply open the editor of your choice (or if you didn't take my advice, the development environment of your choice), create a new file, enter the lines of source code provided in Listing 2.1, and then save the file under the name *main.js*.

```
function showMessage() {  
    alert('Hello World');  
}
```

Listing 2.1 A Very Simple JavaScript Example That Defines a Function

### Note

JavaScript files have the extension *.js*. Other file extensions are also possible, but the *.js* extension has the advantage that editors, development environments, and browsers directly know what the content is about. You should therefore always save all JavaScript files with the *.js* extension. (By the way, browsers recognize JavaScript files delivered by a web server via the *Content-Type header*, a piece of information that comes with the file from the server.)

Listing 2.1 defines a *function* with the name `showMessage`, which in turn calls another function (with the name `alert`) and passes it the message `Hello World`. The `alert` function is a JavaScript standard function, which we will briefly discuss later in this chapter. Functions in general, however, will be detailed in Chapter 3.

Supplemental Downloads for the Book

This code example and all those to come can also be found in the **Product Supplements** area for the book (see <https://www.rheinwerk-computing.com/5554>). There you can easily download the code and open it in your editor or directly in your browser (although we think that the most effective way to learn is to type the examples yourself, following them step by step).

2.2.3 Embedding a JavaScript File in an HTML File

To use the JavaScript source code within a web page, you need to link the JavaScript file to the web page or embed the JavaScript file in the HTML file. This is done via the HTML element named `<script>`.

This element can be used in two different ways: On the one hand, as we will demonstrate subsequently, external JavaScript files can be included in the HTML. On the other hand, JavaScript source code can be written directly between the opening `<script>` tag and the closing `</script>` tag.

An example of the latter method will be shown later, but this approach is only useful in exceptional cases because JavaScript code and HTML code are then mixed—that is, stored in one file (which is not a best practice for the reasons already mentioned). So let's first look at how to do it properly and include a separate file.

The `<script>` element has a total of six attributes, out of which the `src` attribute is certainly the most important one: it's used to specify the path to the JavaScript file that is to be included. (Table 2.2 shows an overview of what the other attributes do.)

Attribute	Meaning	Comment
async	Specifies whether the linked JavaScript file should be downloaded in an asynchronous way in order not to interrupt the download of other files (Section 2.2.5). This only makes sense in combination with the <code>src</code> attribute.	Optional

Table 2.2 The Attributes of the `<script>` Element

Attribute	Meaning	Comment
charset	Specifies the character set of the source code that is embedded via the <code>src</code> attribute. This only makes sense in combination with the <code>src</code> attribute, but is rarely used because most browsers do not respect this attribute. It is also considered better style to use UTF-8 everywhere within a website and define this in the <code>&lt;meta&gt;</code> element via the <code>charset</code> attribute.	Optional
defer	Specifies whether to wait to execute the linked JavaScript file until the web page content has been completely processed (Section 2.2.5). This only makes sense in combination with the <code>src</code> attribute, but is not always supported, especially not by older browsers.	Optional
language	Originally intended to indicate the version of JavaScript code used, but largely ignored by browsers.	Outdated
src	Specifies the path to the JavaScript file to be embedded.	Optional
type	Used to specify the MIME type (see box ahead) in order to identify the scripting language (in our case, JavaScript). However, you can also omit this attribute because <code>text/javascript</code> is used by default, which is supported by most browsers.	Optional

Table 2.2 The Attributes of the `<script>` Element (Cont.)

Now create an HTML file named *index.html* and insert the content shown in Listing 2.2.

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Example</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body>
<!--Here the JavaScript file will be included -->
<script src="scripts/main.js"></script>
</body>
</html>
```

Listing 2.2 Embedding JavaScript in HTML

If you now open this HTML file in the browser, nothing will happen yet because the function we defined in Listing 2.1 is not yet called at any point. Therefore, add the `showMessage()` call at the end of the JavaScript file, as shown in Listing 2.3, and reload the web page in the appropriate browser. Then a small hint dialog should open, containing the



message Hello World and with a slightly different appearance depending on the browser (see Figure 2.12).

```
function showMessage() {
  alert('Hello World');
}
showMessage();
```

Listing 2.3 Function Definition and Function Call

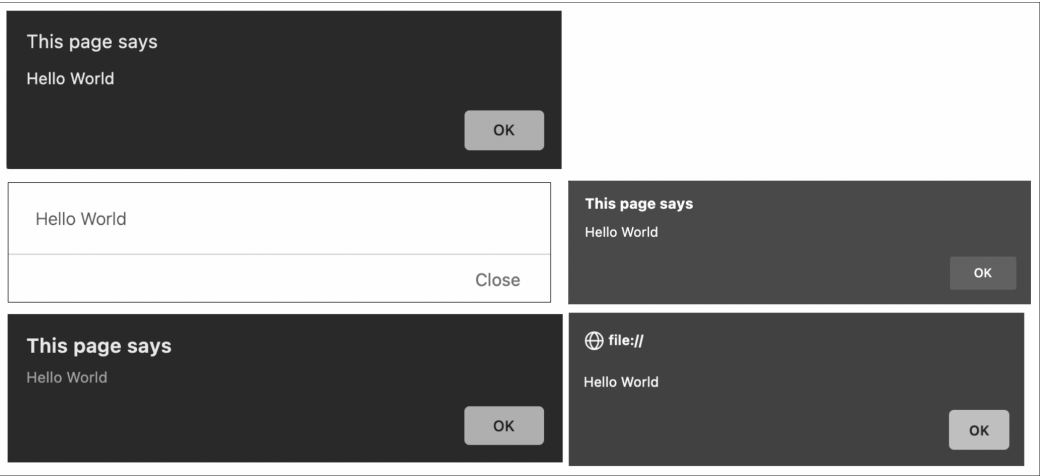


Figure 2.12 Hint Dialogs in Different Browsers

Definition

Multipurpose Internet Mail Extension (MIME) types, also called *internet media types* or *content types*, were originally intended to distinguish between content types within emails containing different content (such as images, PDF files, etc.). Now, however, MIME types are not only used in the context of email, but also whenever data is transmitted over the internet. If a server sends a file with a special MIME type, the client (e.g., the browser) knows directly what type of data is being transmitted.

For JavaScript, the MIME type wasn't standardized for a long time, so there were several MIME types—for example, `application/javascript`, `application/ecmascript`, `text/javascript` and `text/ecmascript`. Since 2006, however, there is an official standard ([www.rfc-editor.org/rfc/rfc4329.txt](http://www.rfc-editor.org/rfc/rfc4329.txt)) that defines the acceptable MIME types for JavaScript. According to this standard, `text/javascript` and `text/ecmascript` are both deprecated, and `application/javascript` and `application/ecmascript` should be used instead. Ironically, it's safest not to specify any MIME type for JavaScript at all (in the `<script>` element) as the type attribute is ignored by most browsers anyway.

Embedding Multiple JavaScript Files

Of course, you can embed several JavaScript files within one HTML file. Simply use a separate `<script>` element for each file you want to include.

2.2.4 Defining JavaScript Directly within the HTML

For the sake of completeness, we'll also show how you can define JavaScript directly within an HTML file. While this is usually not advisable because it means mixing HTML and JavaScript code in one file, it won't hurt to know that it still works.

Simply write the relevant JavaScript code inside the `<script>` element instead of linking it via the `src` attribute. Listing 2.4 shows the same example as in the previous section, but it doesn't use a separate JavaScript file for the JavaScript code. Instead, it embeds the code directly in the HTML. The `src` attribute is therefore omitted completely.

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Example</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body>
<script>
  function showMessage() {
    alert('Hello World');
  }
  showMessage();
</script>
</body>
</html>
```

Listing 2.4 Only Makes Sense in Exceptional Cases: Definition of JavaScript Directly in an HTML File

Note

Note that `<script>` elements that use the `src` attribute must not contain any source code between `<script>` and `</script>`. If there is any, this source code will be ignored.

**Tip**

Use separate JavaScript files for your source code instead of writing it directly into a `<script>` element. This creates a clean separation between the structure (HTML) and the behavior (JavaScript) of a web page.

**The `<noscript>` Element**

You can use the `<noscript>` element to define an HTML section that is displayed when JavaScript is not supported in the browser or has been disabled by the user (see Listing 2.5). However, if JavaScript is supported or enabled, the content of the `<noscript>` element will not be shown.

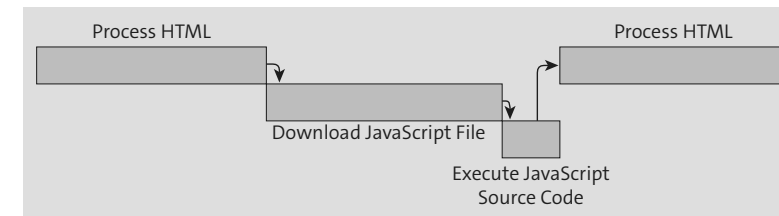
```
<noscript>
  JavaScript is not available or is disabled. <br />
  Please use a browser that supports JavaScript,
  or enable JavaScript in your browser.
</noscript>
```

**Listing 2.5** Example of the Use of the `<noscript>` Element

**2.2.5 Placement and Execution of the `<script>` Elements**

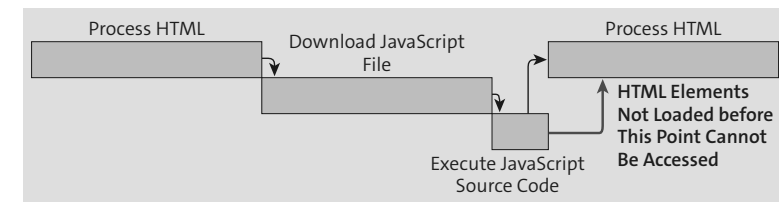
If you had asked a web developer a few (many) years ago where to place a `<script>` element within a web page, they probably would have advised placing it in the `<head>` area of the web page. In the early days of web development, people thought that linked files such as CSS files and JavaScript files should be placed in a central location within the HTML code.

Since then, however, this idea has been abandoned. While CSS files are still placed in the `<head>` area, JavaScript files should be included before the closing `</body>` tag instead. The reason is this: when the browser loads a web page, it loads not only the HTML code but also embedded files such as images, CSS files, and JavaScript files. Depending on processor performance and memory usage, modern browsers are capable of downloading several such files in parallel. However, when the browser encounters a `<script>` element, it immediately starts processing the corresponding source code and evaluating it using the JavaScript interpreter. To be able to do this, the corresponding JavaScript source code must first be downloaded entirely. While this is happening, the browser pauses downloading all other files and *parsing* (i.e., processing) the HTML code, which in turn leads to the user impression that it takes longer to build the web page (see Figure 2.13).

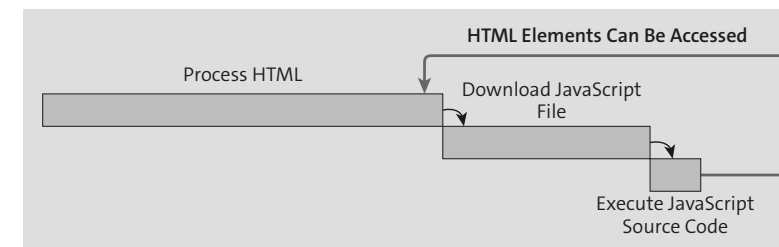


**Figure 2.13** By Default, HTML Code Processing Stops when the Browser Encounters a `<script>` Element

In addition, you will often want to access HTML elements on a web page within the JavaScript source code. (You'll see how this works in Chapter 5.) If the JavaScript code is executed before these HTML elements have been processed, you'll encounter an access error (see Figure 2.14). If you place the `<script>` element before the closing `</body>` tag, though, you are on the safe side in this regard (see Figure 2.15), because in that case all elements included inside the `<body>` element are already loaded (with the exception of other `<script>` elements, of course).



**Figure 2.14** If JavaScript Accesses HTML Elements That Have Not Yet Been Loaded, an Error Occurs

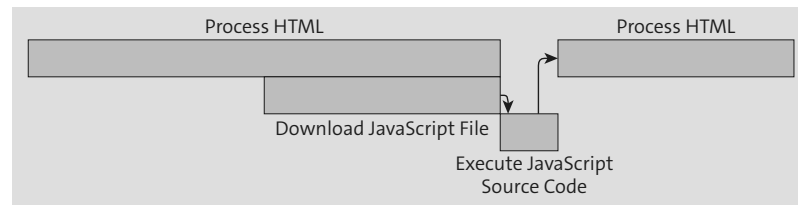


**Figure 2.15** If the `<script>` Element Is Placed before the Closing `</body>` Tag, All Elements inside the `<body>` Element Are Loaded

**Note**

As a rule, you should position `<script>` elements at the end of the `<body>` element. This is because the browser first evaluates the JavaScript source code contained or embedded in each `<script>` element before continuing to load other HTML elements.

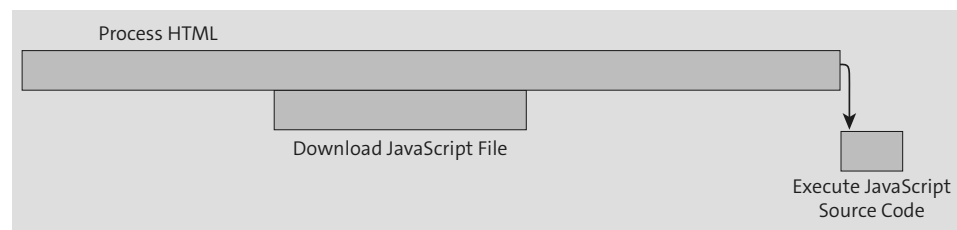
Two attributes that can be used to influence the loading behavior of JavaScript are the `async` and `defer` attributes, which we already mentioned briefly (see Table 2.2). The former ensures that the processing of HTML code is not paused when the browser encounters a `<script>` element. The JavaScript file is downloaded asynchronously (hence the name `async`). This concept is shown in Figure 2.16.



**Figure 2.16** Due to the `async` Attribute, the HTML Code Continues To Be Processed until the Corresponding JavaScript Has Been Downloaded

As you can see, the JavaScript code is executed right away as soon as the corresponding JavaScript file has been completely downloaded.

The `defer` attribute takes this one step further. On the one hand, just like `async`, this attribute ensures that the HTML code processing is not paused. On the other hand, the JavaScript source code is executed only after the HTML code has been fully processed (see Figure 2.17). The execution of the JavaScript code is effectively deferred (hence the name `defer`).



**Figure 2.17** The `defer` Attribute Ensures That the Corresponding JavaScript Is Only Executed after the Entire HTML Code of the Web Page Has Been Loaded

So when should you use which attribute? For now, you can bear in mind that it's probably best not to use either attribute by default. The `async` attribute is only suitable for scripts that work completely independently and have nothing to do with the HTML on the web page. An example of this is the use of Google Analytics. The `defer` attribute, on the other hand, is currently not supported by all browsers, so you should also consider its use with caution.

### Definition

Another way to ensure that all the content of the web page has been loaded before JavaScript code is executed is to use *event handlers* and *event listeners*. We'll introduce both of them in detail in Chapter 6. But for now, we'll show you roughly how both of them are used because they appear in the source code examples in the book before we get to the examples for Chapter 6.

In general, both event handlers and event listeners are used to respond to certain events that occur during the execution of a program and to execute certain code. (There is a small, subtle difference between event handlers and event listeners, but it's not important for now, and we'll explain it in Chapter 6.) Events can be mouse clicks, keystrokes, window resizing actions, and more. For web pages, too, there are various events that are triggered and can be answered by such event handlers and event listeners. For example, an event is triggered when the content of a web page is fully loaded.

To define an event handler for this event, you can use the `onload` attribute: The code you specify here as the value for such an attribute is invoked when the web page is fully loaded. As a value, you can specify a JavaScript *statement*, such as the call to a function, as shown in Listing 2.6.

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Example</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body onload="showMessage()">
  <script src="scripts/main.js"></script>
</body>
</html>
```

### Listing 2.6 Using an Event Handler

Event listeners, however, cannot be defined via HTML. Instead, you use the `addEventListener()` function of the document object (more on this later), to which you pass the name of the event and the function to be executed when the event is triggered (see Listing 2.7).

```
function showMessage() {
  alert('Hello World');
}
document.addEventListener('DOMContentLoaded', showMessage);
```

### Listing 2.7 Using Event Listeners

The `showMessage()` call you just added to the end of the *main.js* file will need to be removed again in both cases. Otherwise, the function will be called twice (once by the script itself and once by the event handler/event listener), and as a consequence a message dialog will be displayed twice in succession.

2.2.6 Displaying the Source Code

All browsers usually provide a way to view the source code of a web page. This can be helpful in many cases—for example, if you want to check how a particular feature is implemented on a website you have discovered.

In Chrome, you can view the source code by following menu path **View • Developer • View Source** (see Figure 2.18); in Firefox, **Tools • Browser Tools • Page Source** (see Figure 2.19); in Safari, **Develop • Show Page Source** (see Figure 2.20); in Opera, **Developer • View Source** (see Figure 2.21); and in Microsoft Edge, **Tools • Developer • View Source** (see Figure 2.22).

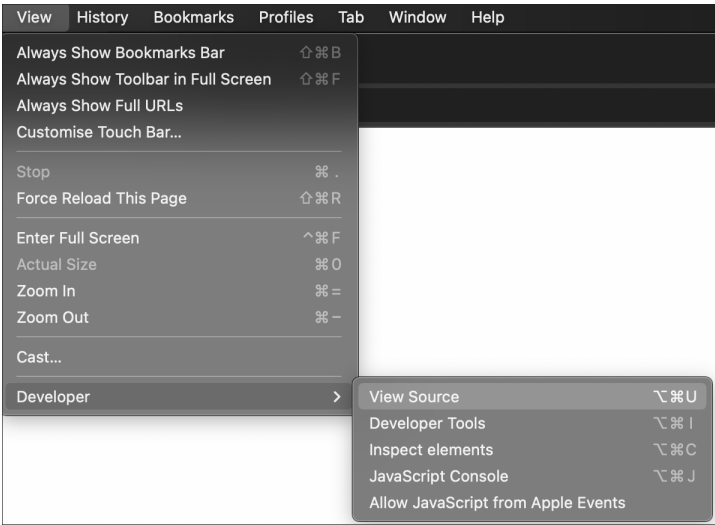


Figure 2.18 Show Source Code in Chrome

**Source Code for More Complex Web Pages**

If you look at the source code of more complex web pages, it's often very confusing. This is usually due to multiple reasons: on the one hand, content is often generated dynamically, and on the other, JavaScript is often deliberately compressed and obscured by web developers—the former to save space, the latter to protect the source code from prying eyes. This book does not deal with the compression and obfuscation of source code.

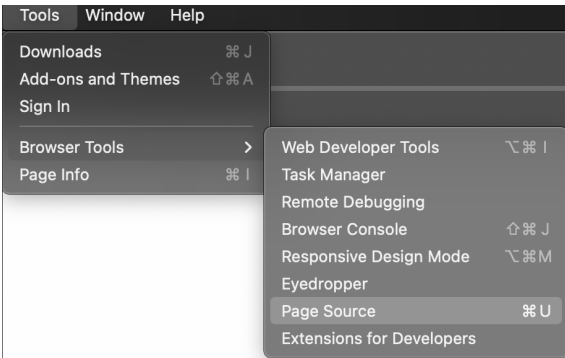


Figure 2.19 Show Source Code in Firefox

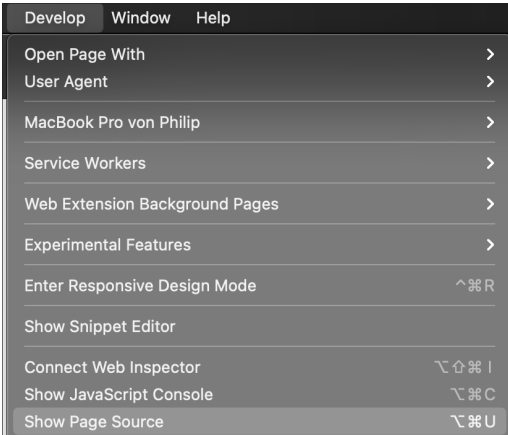


Figure 2.20 Show Source Code in Safari

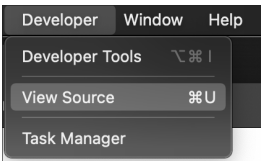


Figure 2.21 Show Source Code in Opera

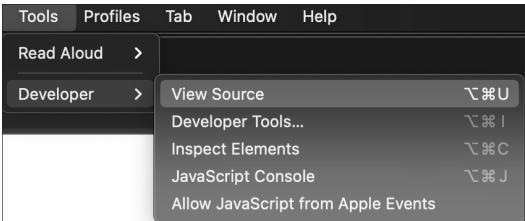


Figure 2.22 Show Source Code in Microsoft Edge



If you display the source code of a web page (no matter in which browser), you are first presented with the corresponding HTML code of the web page. Conveniently, however, embedded files such as CSS files or JavaScript files are linked in this source code view (see Figure 2.23) so that you can easily get to the source code of the linked file as well (see Figure 2.24).

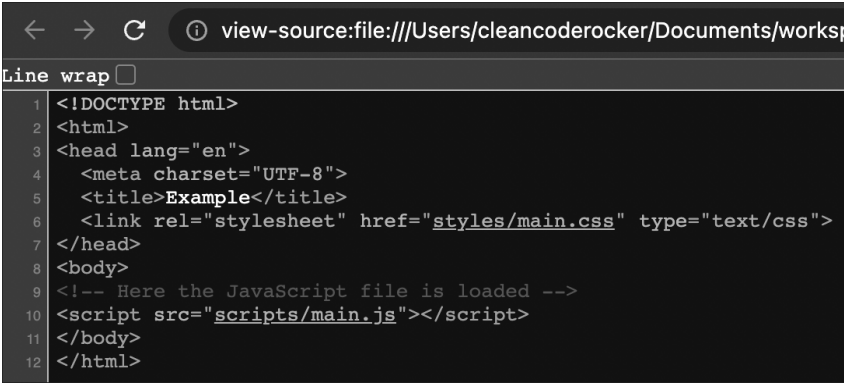


Figure 2.23 Source Code View for HTML in Chrome



Figure 2.24 Source Code View for JavaScript in Chrome

## 2.3 Creating Output

In the *Hello World* example, you have already seen how you can create simple output by calling the `alert()` function. However, there are several other options as well.

### 2.3.1 Showing the Standard Dialog Window

In addition to the already known hint dialog displayed by calling the `alert()` function (see Figure 2.25), the JavaScript language provides two more standard functions for displaying dialog boxes. The first one is the `confirm()` function. It's used to display *confirmation dialogs*—that is, yes/no decisions (see Figure 2.26). In contrast to the hint dialog, the confirmation dialog contains two buttons: one to confirm and one to cancel the corresponding message. The second one is the `prompt()` function. This function opens an *input dialog* where users can enter text (see Figure 2.27).



Figure 2.25 Simple Hint Dialog

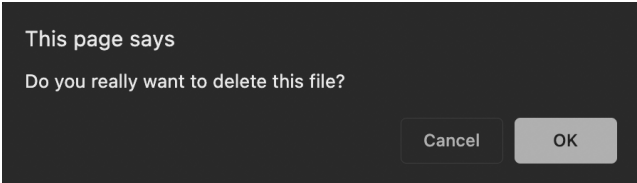


Figure 2.26 Simple Confirmation Dialog

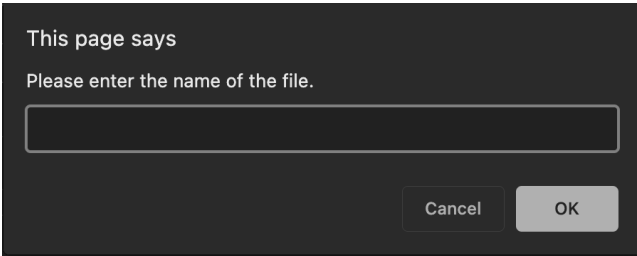


Figure 2.27 Simple Input Dialog

In practice, however, these standard dialogs for hints, confirmation, and input are rarely used because they offer limited options for statements and—as already shown for the hint dialog—their design relies on the layout of the browser being used, which usually does not match the layout of the web page.

For this reason, web developers like to resort to one of the various JavaScript libraries that offer fancier and more functional dialogs (see Figure 2.28). One of these libraries is jQuery UI, which builds on the well-known jQuery library and extends it with various UI components. We will take a closer look at the main library, jQuery, as well as jQuery UI in Chapter 10.



Figure 2.28 Custom Confirmation Dialog with JavaScript

2.3.2 Writing to the Console

When developing JavaScript applications, you'll often want to generate output for yourself for testing purposes only—for example, to return an intermediate result. For such test-only output, it obviously doesn't make sense to present it in dialogs that users would get to see as well. For this reason, all current browsers now offer a *console*, which is suitable for exactly such purposes and which you can access within a JavaScript program in order to output messages. By default, this console is hidden because users of a web page usually can do little with it.

Displaying the Console

To activate the console, proceed as follows, depending on your browser (we won't provide screenshots at this point as the menu items can be found in similar places as the menu items for displaying the source code, mentioned earlier in this chapter):

- In Chrome, select **View • Developer • JavaScript Console**, which opens the console within the Chrome DevTools.
- In Firefox, open the console via **Tools • Browser Tools • Browser Console**.
- In Safari, open the console via **Develop • Show JavaScript Console**.
- In Opera, you must first select **Developer • Developer Tools** and then the **Console** tab.
- In Microsoft Edge, open the console via **Tools • Developer • JavaScript Console**.

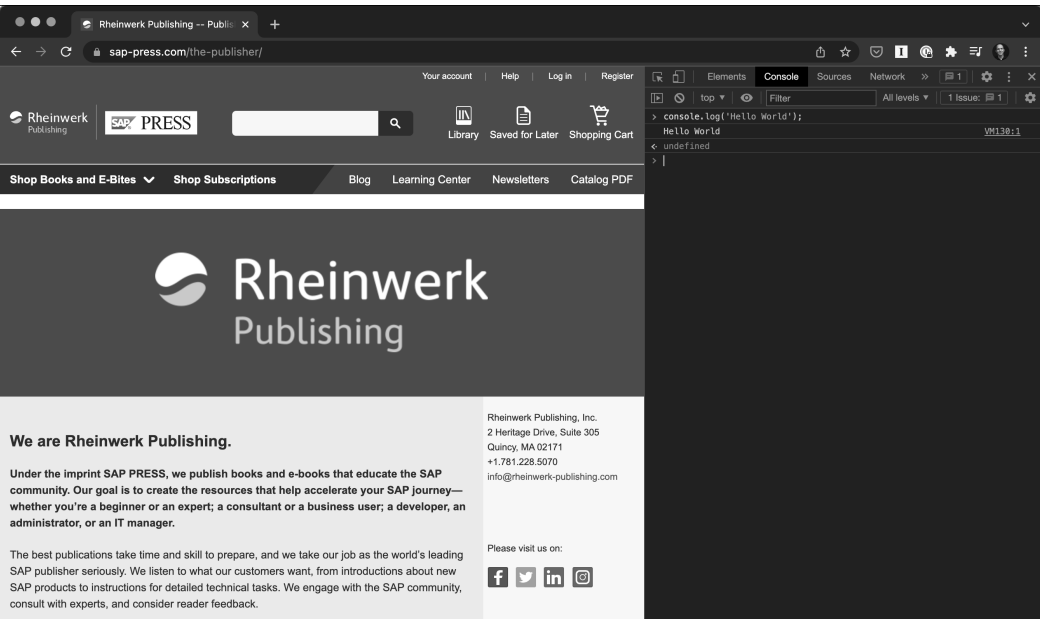


Figure 2.29 By Default, the Console Is Displayed on the Right or at the Bottom of the Browser Window (Google Chrome in This Case)

Figure 2.29 shows the console in the Chrome browser, for example. As you can see, it doesn't really look special, but it will be one of your main tools if you want to use JavaScript for web development. In addition to receiving output, you can also enter your input via the console (more about this in a few moments). In essence, the console is a kind of terminal (or command prompt, if you're a Windows user) that lets you issue JavaScript commands that are then executed in the context of the web page loaded.

Writing Output to the Console

For writing to the console, browsers provide the `console` object. This is a JavaScript object first introduced by the Firefox plug-in named Firebug (<https://getfirebug.com>), and it provides various ways to generate output to the console. Firebug itself has been discontinued, but the `console` object (although still not included in the ECMAScript standard) is available in almost every JavaScript runtime environment.

**Standardized API for Working with the Console**

The individual methods provided by the `console` object differ from runtime environment to runtime environment. To counteract this, there are efforts underway to create a standardized API.

A generally supported method is the `log()` method, which can be used to generate simple console output. To try using the console, simply replace the source code of the `main.js` file with the source code in Listing 2.8 and call the web page again.

```
// scripts/main.js
function showMessage() {
  console.log('Hello developer world');
}
```

Listing 2.8 Simple JavaScript Example

Depending on the browser, the result should be similar to that shown in Figure 2.30.

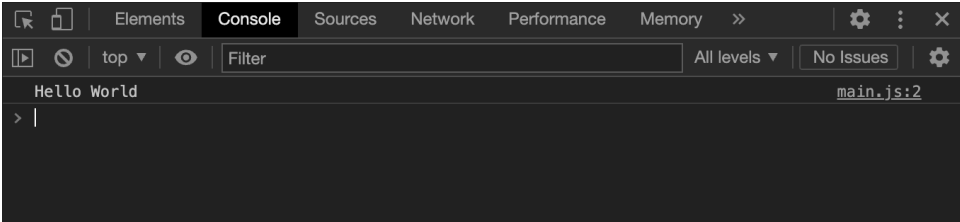


Figure 2.30 Output to the Console in Chrome

In addition to the `log()` method, `console` provides several other methods. An overview of the most important ones is provided in Table 2.3.

Method	Description
<code>clear()</code>	Clears the console.
<code>debug()</code>	Used to output a message intended for <i>debugging</i> (or <i>troubleshooting</i> ). (You may first need to set the appropriate developer tools to return this type of output.)
<code>error()</code>	Used to output an error message. Some browsers display an error icon next to the output message within the console.
<code>info()</code>	This will display an info message in the console. Some browsers—Chrome, for example—also output an info icon.
<code>log()</code>	Probably the most commonly used method of <code>console</code> . Generates normal output to the console.
<code>trace()</code>	Outputs the <i>stack trace</i> —that is, the <i>method call stack</i> (see also Chapter 3) to the console.
<code>warn()</code>	Used to issue a warning to the console. Again, most browsers will display a corresponding icon next to the message.

Table 2.3 Most Important Methods of the console Object

Listing 2.9 shows the corresponding source code for using the `console` object. The output for the individual methods is highlighted with colors or icons, depending on the browser (see Figure 2.31).

```
console.log('Hello developer world');    // Output of a normal message
console.debug('Hello developer world');  // Output of a debug message
console.error('Hello developer world');   // Output of an error message
console.info('Hello developer world');    // Output of an info message
console.warn('Hello developer world');    // Output of a warning
```

Listing 2.9 Using the console Object

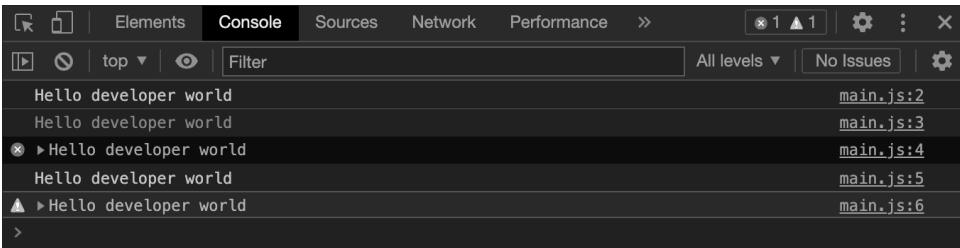


Figure 2.31 Different Message Types Are Highlighted with Colors or Icons

Writing Input to the Console

In the last few screenshots, you may have noticed the `>` sign below the output. Here, you can enter any JavaScript code and have it executed right away. This is a great way to quickly test simple scripts, and actually indispensable for web development. Try it: type the `showMessage()` command in the prompt and then press the `Enter` key to execute the command. The results are displayed in Figure 2.32.

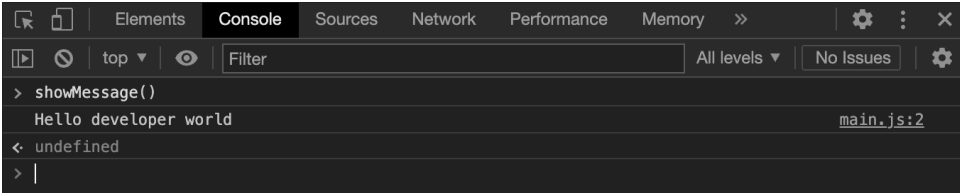


Figure 2.32 You Can Also Execute Source Code via the Console

**Note**  
The console window and the `console` object are important tools for web developers. Make yourself familiar with both when you get a chance.

**Logging Libraries**  
The `console` object works well for quick output during development. However, if a web page goes live or a JavaScript application is used in production, you don't really want to use `console` calls any longer (even though they are usually not displayed to the user). In practice, you often use special *logging libraries* that enable console output to be activated (for development) but also deactivated again (for productive use) via specific configuration settings. To start, and also for the examples in this book, however, the use of the `console` object should be sufficient.

2.3.3 Using Existing UI Components

Because the use of `alert()`, `confirm()`, and `prompt()` is rather outdated and only useful for quick testing, and the output via the `console` object is reserved for developers anyway, you obviously still need a way to create an appealing output for the user of a web page. To this end, you can write the output of a program into existing UI components such as text fields and the like.

Listing 2.10, Listing 2.11, and Figure 2.33 show an example. It consists of a simple form that can be used to determine the result of adding two numbers. The two numbers can be entered into two text fields, the addition is triggered by pressing the button, and the result is written into the third text field.

You don't need to understand the code for this example yet, and we won't into the details at this point. For now, just keep in mind that when developing for the web with JavaScript, it's relatively common to use HTML components for sending output from a program to the user.

```
// scripts/main.js
function calculateSum() {
  const x = parseInt(document.getElementById('field1').value);
  const y = parseInt(document.getElementById('field2').value);
  const result = document.getElementById('result');
  console.log(x + y);
  result.value = x + y;
}
```

Listing 2.10 The JavaScript Code of the main.js File

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Example</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body>
<div class="container">
  <div class="row">
    <label for="field1">X</label> <input id="field1" type="text" value="5">
  </div>
  <div class="row">
    <label for="field2">Y</label> <input id="field2" type="text" value="5">
  </div>
  <div class="row">
    <label for="result">Result: </label> <input id="result" type="text">
    <button onclick="calculateSum()">Calculate sum</button>
  </div>
</div>
<script src="scripts/main.js"></script>
</body>
</html>
```

Listing 2.11 The HTML Code for the Example Application

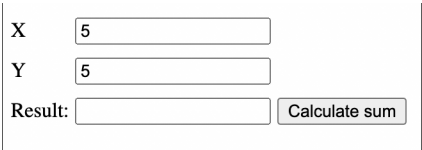


Figure 2.33 Example Application

**DOM Manipulation**

The most complex scenario is when you dynamically modify a web page to produce output—for example, dynamically modify a table to display tabular structured data. We will discuss this topic of DOM manipulation in more detail in Chapter 5.

## 2.4 Summary

In this chapter, you learned how to create JavaScript files and embed them in HTML. You now have the basic knowledge for executing the examples in the next chapters. The following key points were presented in this chapter:

- Three languages are important for frontend development: HTML as a *markup language* to define the structure of a web page, CSS as a *style language* to define design and layout, and JavaScript as a *programming language* to add additional behavior and interactivity to a web page.
- You can specify JavaScript directly using the `<script>` element or can embed a separate JavaScript file using the `src` attribute of the `<script>` element. We recommend the latter, as it ensures a clean separation between the structure (HTML) and behavior (JavaScript) of the web page.
- You should always place `<script>` elements before the closing `</body>` tag, as this ensures that the web page content is fully loaded.
- JavaScript inherently provides three functions for generating output: `alert()` for creating hint dialogs, `confirm()` for creating confirmation dialogs, and `prompt()` for creating input dialogs.
- In practice, however, instead of these (more or less obsolete) functions, people use fancier dialogs, such as those offered by the jQuery library.
- In addition, all current browsers provide the possibility to generate output via a console, which is primarily intended for you to use as a developer.



## Chapter 10

# Simplifying Tasks with jQuery

*Many tasks that can now be performed relatively easily with JavaScript were for a long time only possible with a relatively large amount of source code due to browser differences. For this reason, various libraries have emerged to simplify different tasks such as working with the DOM. One of the most famous of these libraries is jQuery, which even today is part of every web developer's toolbox.*

Probably one of the best-known JavaScript libraries is the jQuery library (<https://jquery.com>), which, in part, considerably simplifies working with JavaScript. Although many things are now also possible with standard methods of the DOM API, jQuery is still a library to be taken seriously. This chapter provides an overview of using jQuery, including how to simplify accessing and manipulating the DOM, working with events, and formulating Ajax requests.

### Note

The jQuery library is so extensive that we can't cover all its aspects in one chapter. Instead, we'll offer a selection of topics that are representative and give a good introduction to the library. Also, we won't discuss the selected topics in great detail but will describe the code examples relatively concisely (we're assuming that you've already acquired the necessary basic knowledge, such as DOM processing, events, Ajax, and so on, throughout the preceding chapters).

## 10.1 Introduction

As you've seen in the previous chapters, there are differences between different browsers with regard to DOM manipulation, event processing, and Ajax. The jQuery library abstracts such browser-specific details and provides a unified interface, and not only for the aforementioned topics. So essentially, jQuery has the following advantages:

- **Simplified working with the DOM**

jQuery simplifies access to elements of the DOM tree by providing various helper methods. In Section 10.2, we discuss this topic in more detail. By the way: standard methods of the current DOM API, like `querySelector()` and `querySelectorAll()` (which are not available in older browsers), are based on ideas from jQuery.

■ **Simplified working with events**

jQuery simplifies working with events and provides helper methods for this purpose, which we'll introduce in Section 10.3.

■ **Simplified phrasing of Ajax requests**

jQuery simplifies the phrasing of Ajax requests—again, by hiding browser-specific details. We'll present the corresponding helper methods in Section 10.4.

**jQuery Isn't Always Necessary**

Although jQuery is a really powerful library, you shouldn't make the mistake of equating jQuery with JavaScript and first learning jQuery and then the JavaScript language. jQuery can certainly be of support in many cases, but often the use of the library isn't even necessary because you can already solve the corresponding tasks with pure JavaScript code or even other, leaner libraries. Websites like You Might Not Need jQuery (<http://youmightnotneedjquery.com>) demonstrate this with various examples.

**Tip**

In principle, it's not bad for a JavaScript developer to both be able to use libraries like jQuery and have a firm grasp of the basic language concepts as well.

**10.1.1 Embedding jQuery**

The jQuery library can be embedded in several ways. At <https://jquery.com/download/>, you can download the current version of the library. Besides the "normal" version, a minified (i.e., compressed) version is available for download, which is as small as possible in terms of file size. After you've downloaded the file, you can include it as usual via the `<script>` element (see Listing 10.1).

```
<!DOCTYPE html>
<html>
<head lang="en">
  <title>jQuery example</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body>
  <script src="scripts/jquery-3.6.0.min.js"></script>
  <script src="scripts/main.js"></script>
</body>
</html>
```

**Listing 10.1** Embedding a Downloaded Version of jQuery

**Minified Versions versus Nonminified Versions**

Most libraries offer both a normal (nonminified) version and a minified version for download. In the latter, spaces and often comments within the code are removed, for example, and much more is optimized to reduce the file size and thus reduce download time. Consequently, minified versions of libraries are suitable for use in a production system. The nonminified version is actually only suitable if you also want to take a look at the corresponding library during development—during debugging, for example.

**10.1.2 Embedding jQuery via a Content Delivery Network**

If you download jQuery as described and embed it as a local dependency in your web page and then load your website onto a server, you must also load jQuery onto the appropriate server. Alternatively, you have the option of integrating jQuery via a *content delivery network* (CDN; see note box). For jQuery, the corresponding URL (for the current version of the library) is <https://code.jquery.com/jquery-3.6.0.min.js> (see Listing 10.2).

```
<!DOCTYPE html>
<html>
<head lang="en">
  <title>jQuery example</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script src="scripts/main.js"></script>
</body>
</html>
```

**Listing 10.2** Embedding jQuery via a Content Delivery Network

**Content Delivery Network**

A *content delivery network* (also known as a *content distribution network*) is a network of servers connected via the internet that distribute requests so that they can be answered as quickly as possible. Typically, the geographical location of a request plays a major role: for example, if a user from Germany accesses your web page and you have embedded jQuery there via a CDN URL (<https://code.jquery.com>), the corresponding code is sent to the user from a server in Germany. A user accessing your web page from the US, on the other hand, is served by a server located there.

### 10.1.3 Using jQuery

The core of jQuery is the `jQuery()` function or the equivalent shortcut function `$()` (called the *jQuery method* ahead). This function can be called with various arguments (see <http://api.jquery.com/jquery> for details), three forms of which are used particularly frequently:

- **Call with CSS selector**

In this case, you pass a selector to the jQuery method (similar to the CSS selectors, but more about that in a moment) and receive an object as the return value that contains the elements of the web page that match the selector. Examples of this are shown in Listing 10.3 and Listing 10.4. The object returned by the method represents a so-called wrapper object (called the *jQuery object* ahead) for the corresponding elements and provides various methods for these elements (more on this later).

```
const selectedElements = jQuery('body > div > span');
```

**Listing 10.3** The `jQuery()` Function

```
const selectedElements = $('body > div > span');
```

**Listing 10.4** The More Common Shortcut Function, `$()`

- **Call with nodes from the DOM tree**

As an alternative to calling the jQuery method with a selector, it can also be called with a node of the DOM tree or with the corresponding JavaScript object representing the respective node. Here as well, the jQuery object represents a wrapper object around the passed node and provides additional methods. For example, to define an event listener that is called when the `document` object is fully loaded, proceed as in Listing 10.5. The `ready()` method doesn't exist for the `document` object but is provided indirectly by the jQuery object (see also Section 10.3.2).

```
$(document).ready(() => {
  console.log('Web page loaded');
});
```

**Listing 10.5** Calling the jQuery Method with a Node from the DOM Tree

- **Call with HTML string**

You can also use the jQuery method to create new elements. To do this, simply pass the appropriate HTML code for the element you want to create to the method as a string, as shown in Listing 10.6.

```
const newElement = $('<div>New element</div>');
```

**Listing 10.6** Calling the jQuery Method with an HTML String

#### Note

In all cases shown, the return value of the jQuery method is an object that adds additional functionality, the jQuery object, to the corresponding elements. This object contains references to one or more nodes of the DOM tree, hereafter referred to as *selected nodes* or *selected elements*.

### 10.1.4 Simplifying Tasks with jQuery

The fact that jQuery simplifies working with the DOM is best demonstrated using an example. Suppose you have an HTML list in which each list entry contains a URL (as text, not as a link), and you'd like to use JavaScript to create real links from the URLs at runtime. In other words: The text content of the list entries is to be converted into `<a>` elements.

Using pure JavaScript, you would probably proceed as shown in Listing 10.7. First, the appropriate `<li>` elements must be selected (here, for simplicity, all `<li>` elements of the entire web page). Then, in each case, the text content must be extracted and removed, and a new `<a>` element must be created, its `href` attribute and text content must be set, and the element must be added to the `<li>` element as a child element.

```
'use strict';
function init() {
  const listItems = document.getElementsByTagName('li');
  for(let i=0; i<listItems.length; i++) {
    const listItem = listItems[i];
    const url = listItem.textContent;
    listItem.textContent = '';
    const link = document.createElement('a');
    link.setAttribute('href', url);
    const linkText = document.createTextNode(url);
    link.appendChild(linkText);
    listItem.appendChild(link);
  }
}
document.addEventListener('DOMContentLoaded', init);
```

**Listing 10.7** Creating Links Using Pure JavaScript

Not exactly little code for actually a trivial task. And it doesn't get any better if you use the `innerHTML` property instead of the `createElement()`, `setAttribute()`, and `createTextNode()` DOM methods, as shown in Listing 10.8. This code also looks relatively cluttered and cobbled together.

```
'use strict';
function init() {
  const listItems = document.getElementsByTagName('li');
  for(let i=0; i<listItems.length; i++) {
    listItems[i].innerHTML = '<a href="'
    + listItems[i].textContent + '">'
    + listItems[i].textContent + '</a>';
  }
}
document.addEventListener('DOMContentLoaded', init);
```

Listing 10.8 Creating Links Using innerHTML

With jQuery, things get a little more elegant, to say the least. The corresponding code is shown in Listing 10.9. Here the `wrapInner()` method comes into play, which is made available to the selected elements by the jQuery object. This method wraps the contents of the selected elements with the HTML code returned by the passed function in the example. Much simpler than the previous code!

```
'use strict';
function init() {
  $('li').wrapInner(
    function() {
      return '<a href="' + this.textContent + '"></a>'
    }
  );
}
$(document).ready(init)
```

Listing 10.9 Creating Links Using jQuery

jQuery Plug-ins

By the way, if you don't find a functionality within jQuery, there are thousands of plug-ins available on the internet. A good overview is given by the official jQuery registry at <https://plugins.jquery.com/>.

10.2 Working with the DOM

Working with the DOM wasn't always as comfortable as it is now thanks to methods like `querySelector()` and `querySelectorAll()`. For a long time, jQuery was the first choice when it came to processing the DOM in a relatively simple way—to select,

modify, or add elements, for example. Even today, jQuery supports you in the following tasks, among others:

- Selection of elements (Section 10.2.1)
- Accessing and modifying content (Section 10.2.2)
- Filtering selected elements (Section 10.2.3)
- Accessing attributes (Section 10.2.4)
- Accessing CSS properties (Section 10.2.5)
- Navigating between elements (Section 10.2.6)
- Using effects (Section 10.2.7)

10.2.1 Selecting Elements

Using jQuery, elements can be selected using CSS-like selectors. These selectors can be divided into the following groups:

- **Basic selectors**  
Essential selectors that you already know from CSS (see Table 10.1)
- **Hierarchy selectors**  
Selectors involving the hierarchy of elements (see Table 10.2), also already known from CSS
- **Basic filter selectors**  
Selectors that allow you to more specifically filter individual elements, not all of which exist in CSS (see Table 10.3)
- **Content filter selectors**  
Selectors that include the content of elements (see Table 10.4)
- **Visibility filter selectors**  
Selectors involving the visibility of elements (see Table 10.5)
- **Attribute filter selectors**  
Selectors that include the attributes of elements (see Table 10.6)
- **Form filter selectors**  
Selectors that are specifically useful for selecting form elements (see Table 10.7)
- **Child filter selectors**  
Selectors for selecting child elements (see Table 10.8)

Selector	Description
*	Elements with any element name
elementName	Elements of type elementName

Table 10.1 Basic Selectors in jQuery



Selector	Description
#id	Element with the ID id
.class	Elements of the class class
selektor1, selector2	Elements that match either the selektor1 selector or the selector2 selector

Table 10.1 Basic Selectors in jQuery (Cont.)

Selector	Description
element1 element2	All elements of type element2 that are inside an element of type element1
element1 > element2	All elements of type element2 that are direct child elements of an element of type element1
element1 + element2	All elements of type element2 that directly follow an element of type element1
element1 ~ element2	All elements of type element2 that follow an element of type element1

Table 10.2 Hierarchy Selectors in jQuery

Selector	Description
:animated	Selects elements that are currently used within an animation.
:header	Selects all heading elements—that is, <h1>, <h2>, <h3>, <h4>, <h5>, and <h6>.
:lang()	Selects all elements for the passed language.
:not()	Selects all elements that do not match the passed selector.
:root	Selects the root element (not the document node)—that is, the <html> element.
:target	Selects the element identified by the fragment ID of the corresponding URL. For example, if the URL is <i>http://www.javascripthandbuch.de#jquery</i> , then \$(':target') will select the element with the ID jquery.

Table 10.3 Basic Filter Selectors in jQuery

Selector	Description
:contains()	Selects all elements that contain the passed text
:empty	Selects all elements that have no child elements or child nodes
:has()	Selects all elements that contain at least one element that matches the passed selector
:parent	Selects all elements that have at least one child node

Table 10.4 Content Filter Selectors in jQuery

Selector	Description
:hidden	Selects all elements that are not visible
:visible	Selects all visible elements

Table 10.5 Visibility Filter Selectors in jQuery

Selector	Description
[name = "value"]	Selects elements with the attribute name for which the values are a series of values separated by minus signs and where the first value is value
[name*= "value"]	Selects elements with the attribute name, the value of which contains value as a substring
[name~= "value"]	Selects elements with the attribute name, the value of which is a list of values, one of which is equal to value
[name\$= "value"]	Selects elements with the attribute name, the value of which ends with value
[name= "value"]	Selects elements with the attribute name that has the value value
[name!= "value"]	Selects elements with the attribute name that do not have the value value
[name^= "value"]	Selects elements with the attribute name, the value of which begins with value
[name]	Selects elements with the attribute name
[name= "value"] [name2= "value2"]	Selects elements with the attribute name having the value value and with the attribute name2 having the value value2

Table 10.6 Attribute Filter Selectors in jQuery

Selector	Description
:button	Selects all buttons
:checkbox	Selects all checkboxes
:checked	Selects all selected or activated elements
:disabled	Selects all disabled elements
:enabled	Selects all activated elements
:focus	Selects all elements that have the focus
:file	Selects all file input fields
:image	Selects all elements with the attribute type having the value image
:input	Selects all input fields
:password	Selects all password fields
:radio	Selects all radio buttons
:reset	Selects all elements with the attribute type having the value reset
:selected	Selects all selected elements
:submit	Selects all elements with the attribute type having the value submit
:text	Selects all text input fields

Table 10.7 Form Filter Selectors in jQuery

Selector	Description
:first-child	Selects the first child element
:first-of-type	Selects the first element of a given type
:last-child	Selects the last child element
:last-of-type	Selects the last element of a given type
:nth-child()	Selects the nth child element
:nth-last-child()	Selects the nth child element, counting from the end

Table 10.8 Child Filter Selectors in jQuery

Selector	Description
:nth-of-type()	Selects the nth element of a given type
:nth-last-of-type()	Selects the nth element of a given type, counting from the end
:only-child	Selects elements that are the only child element of their parent element
:only-of-type()	Selects elements that are the only child element of their parent element of a given type

Table 10.8 Child Filter Selectors in jQuery (Cont.)

You can find some examples of using these selectors in Listing 10.10, and complete lists of all corresponding selectors can be found in the tables ahead.

```
$(document).ready(() => {
  const inputElements = $('input');           // all <input> elements
  const john = $('#john');                   // element with the ID "john"
  const oddElements = $('.odd');              // elements of the class "odd"
  const elements = $('td, th');              // all <td>- und <th> elements
  const inputJohn = $('input[name="john"]');  // <input> elements the
                                              // name attribute of which
                                              // has the value "john"
  const oddRows = $('tr').odd();              // all "odd" <tr> elements
  const evenRows = $('tr').even();            // all "even" <tr> elements
  const listItemsAtIndex = $('li:eq(2)');     // all <li> elements at index 2
  const allOthers = $(':not(li)');            // all elements other than <li>
  const notExample = $(':not(.example)');     // all elements without the CSS
                                              // class "example"
});
```

Listing 10.10 A Few Examples of Using Selectors

Note

Some selectors such as `:odd` (for selecting all "odd" elements), `:even` (for selecting all "even" elements), `:first` (for selecting the first elements), and `:last` (for selecting the last elements) are deprecated since jQuery 3.4. Instead, you should first select the elements using the appropriate selector and apply the `odd()`, `even()`, `first()`, and `last()` methods on the result set (see Listing 10.10 for the use of `odd()` and `even()`).

Combination of Selectors

The individual selectors naturally can be combined with each other, as you know from CSS.

10.2.2 Accessing and Modifying Content

After you’ve selected elements via the jQuery method using the selectors presented in the previous section, the jQuery object provides various methods for the selected elements to access and modify the content. These include but are not limited to the following:

- Accessing and modifying HTML and text content (see Table 10.9)
- Adding content within an element (see Table 10.10 and Figure 10.1)
- Adding content outside an element (see Table 10.11 and Figure 10.1)
- Adding content around an element (see Table 10.12)
- Replacing content (see Table 10.13)
- Removing content (see Table 10.14)

Method	Description
html()	Without an argument, this method returns the HTML content of an element. With an argument, this method sets the HTML content of an element.
text()	Without an argument, this method returns the text content of an element. With an argument, this method sets the text content of an element.

Table 10.9 Methods for Retrieving and Defining Content

Method	Description
append()	Adds content to the end of the selected elements: <code>\$(a).append(b)</code> adds content <i>b</i> to the end of element <i>a</i> (see Figure 10.1).
appendTo()	Opposite of <code>append()</code> ; that is, <code>\$(a).appendTo(b)</code> adds element <i>a</i> as content to the end of element <i>b</i> .
prepend()	Inserts content at the beginning of the selected elements: <code>\$(a).prepend(b)</code> adds content <i>b</i> to the beginning of element <i>a</i> (see Figure 10.1).
prependTo()	Opposite of <code>prepend()</code> ; that is, <code>\$(a).prependTo(b)</code> adds element <i>a</i> as content to the beginning of element <i>b</i> .

Table 10.10 Methods for Adding Content within an Element

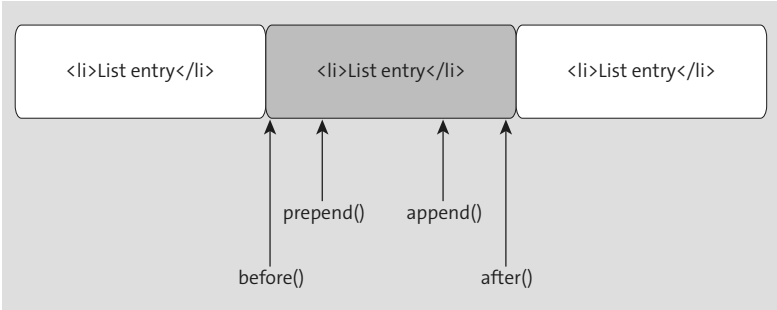


Figure 10.1 The Different Methods for Adding Content inside and outside Elements

Method	Description
after()	Adds content after each of the selected elements: <code>\$(a).after(b)</code> inserts content <i>b</i> after element <i>a</i> (see Figure 10.1).
before()	Adds content before each of the selected elements: <code>\$(a).before(b)</code> inserts content <i>b</i> before element <i>a</i> (see Figure 10.1).
insertAfter()	Opposite of <code>after()</code> ; that is, <code>\$(a).insertAfter(b)</code> inserts element <i>a</i> after element <i>b</i> .
insertBefore()	Opposite of <code>before()</code> ; that is, <code>\$(a).insertBefore(b)</code> inserts element <i>a</i> before element <i>b</i> .

Table 10.11 Methods for Adding Content outside an Element

Method	Description
clone()	Creates a copy of the selected elements. More precisely, a so-called deep copy is created, which also copies child elements of the selected elements.
wrap()	Adds new content around each of the selected elements.
wrapAll()	Adds new content around all selected elements.
wrapInner()	Adds new content around the content of each of the selected elements.

Table 10.12 Methods for Adding Content around an Element

Method	Description
replaceWith()	Replaces the selected elements with new content: <code>\$(a).replaceWith(b)</code> replaces element <i>a</i> with content <i>b</i> .

Table 10.13 Methods for Replacing Content

Method	Description
replaceAll()	Opposite of replaceWith(); that is, \$(a).replaceAll(b) replaces all elements selected by selector b with the content in a.

Table 10.13 Methods for Replacing Content (Cont.)

Method	Description
detach()	Removes the selected elements from the DOM tree but retains references to the removed elements so that they can be reincorporated into the DOM tree at a later time
empty()	Removes all child nodes from the selected elements
remove()	Removes the selected elements from the DOM tree
unwrap()	Removes the parent element from each of the selected elements

Table 10.14 Methods for Removing Content

Some examples of these methods are shown in Listing 10.11. For example, you can use the `html()` method to access the HTML content, the `text()` method to access the text content (both read and write access), `append()` to append new content to the existing content of the selected elements, `prepend()` to insert content before the existing content, `after()` to append new content to the selected elements, and `before()` to insert content before the selected elements.

```
// Add new HTML content
$('#main').html('<div>New content</div>');
// Access the HTML content
const htmlContent = $('#main').html();

// Add new text content
$('#main').text('New text content');
// Access the text content
const textContent = $('#main').text();

// Add new content after the
// existing content of each <div> element
// with the CSS class "example"
$('div.example').append('<p>Example</p>');

// Add new content before the
// existing content of each <div> element
// with the CSS class "example"
$('div.example').prepend('<p>Example</p>');
```

```
// Add new content after each
// <div> element with the CSS class "example"
$('div.example').after('<p>Example</p>');

// Add new content before each
// <div> element with the CSS class "example"
$('div.example').before('<p>Example</p>');
```

Listing 10.11 Some Examples of Changing and Adding Content

10.2.3 Filtering Selected Elements

The jQuery object also provides various methods for the selected elements, which you can use to further narrow down the currently selected elements (see Table 10.15). Some examples are shown in Listing 10.12: for example, the `eq()` method limits the selection to the element at index 2 (i.e., the third `<li>` element in the example); the `first()` and `last()` methods let you select the first and last of the currently selected elements, respectively; the `filter()` method limits the selection to the specified selector; and the `not()` method limits the selection to the elements that do not match the specified selector. You can also use the `has()` method to select the elements that have at least one child element that matches the given selector. Figure 10.2 illustrates the filter methods shown in Listing 10.12.

```
// Selection of the third <li> element
$('li').eq(2);
// Selection of the first <li> element
$('li').first();
// Selection of <li> elements that have the CSS class ".selected"
$('li').filter('.selected');
// Selection of all <li> elements that contain a <ul> element
$('li').has('ul');
// Selection of all elements that have the CSS class ".selected"
$('li').has('.selected');
// Selection of the last <li> element
$('li').last();
// Selection of all class attributes of the <li> elements
$('li').map(() => { this.className });
// Selection of all <li> elements that do not have the CSS class ".selected"
$('li').not('.selected');
// Selection of the first two <li> elements
$('li').slice(0, 2);
```

Listing 10.12 Usage of Different Filter Methods

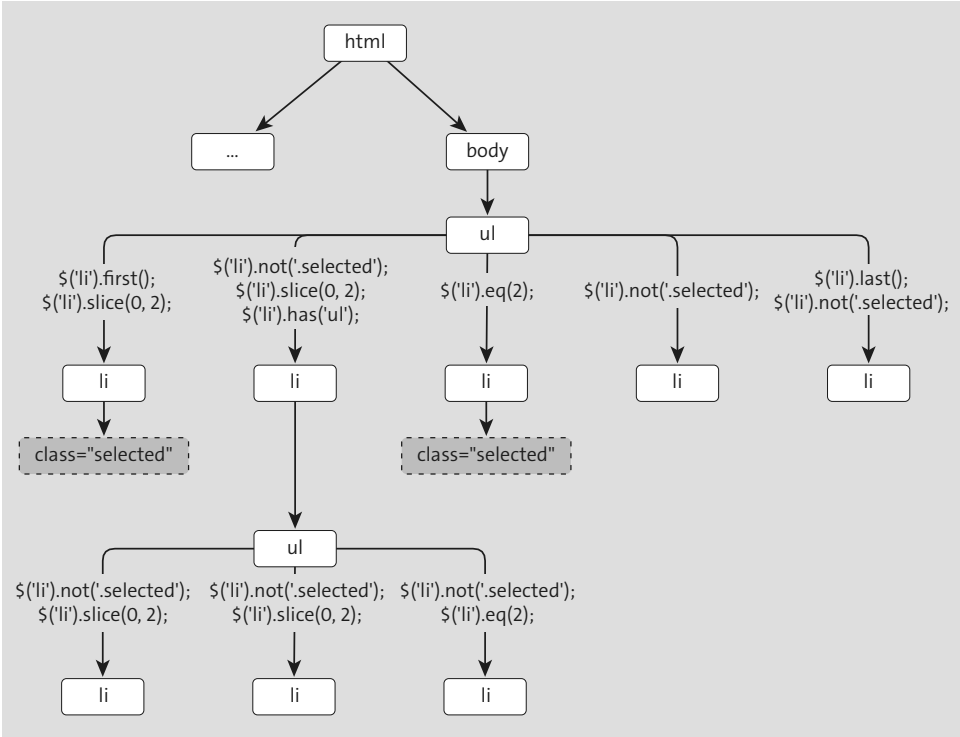


Figure 10.2 Narrowing Down Elements via jQuery Filter Methods

Method	Description
add()	Adds new elements to a selection of elements
addBack()	Adds a previous selection to the current selection of elements
eq()	Reduces the selected elements to one element at a given index
filter()	Reduces the selected elements to those elements (1) that match the passed selector or (2) for which the passed filter function returns true
find()	Selects the child elements of the selected elements that (1) match the passed selector, (2) are contained in the passed jQuery object, or (3) are equal to the passed element
first()	Reduces the selected elements to the first element
has()	Reduces the selected elements to those elements that have a child element that (1) matches the passed selector or (2) is equal to the passed element

Table 10.15 Methods for Filtering

Method	Description
is()	Checks if at least one of the selected elements (1) matches the passed selector, (2) returns true for the passed filter function, (3) is contained in the passed jQuery object, or (4) is one of the elements passed as parameter
last()	Reduces the selected elements to the last element
not()	Reduces the selected elements to those that (1) do not match the passed selector, (2) do not match the passed filter function, or (3) are not contained in the passed jQuery object
slice()	Reduces the selected elements to a subset defined by start and end index

Table 10.15 Methods for Filtering (Cont.)

10.2.4 Accessing Attributes

The jQuery object also provides some methods to access HTML attributes (see Table 10.16). The `attr()` method can be used to determine or reset values of attributes, as shown in Listing 10.13: if you pass the name of an attribute to the method, the method returns the corresponding value of the attribute; if, on the other hand, you pass another string as the second argument to the method, this string is used as the new value for the attribute.

Alternatively, you can pass an object to the method in order to add several attributes in one go. In this case, the object's property names are used as names for the attributes, and the object property values are used as values for the attributes.

To delete attributes, however, use the `removeAttr()` method. For adding and removing CSS classes, the methods `addClass()` and `removeClass()` are available. However, these two methods are actually redundant—at least for newer browsers that support the `classList` property, which provide equivalent functionality via the `add()` and `remove()` methods.

```
const element = $('#main');
// Read access to the "href" attribute of the element
const href = element.attr('href');
// Write access to the "href" attribute of the element
element.attr('href', 'index.html');
// Alternative write access via configuration object
element.attr({
  href: 'index.html',
  target: '_blank'
});
// Remove the "href" attribute from the element
element.removeAttr('href');
```



```
// Add a CSS class
element.addClass('highlighted');
// Remove a CSS class
element.removeClass('highlighted');
```

Listing 10.13 Access to Attributes and CSS Classes

Method	Description
attr()	With one argument, this method returns the value for an attribute (e.g., <code>\$('#a#main').attr('href')</code> ). With two arguments, this method sets the value of an attribute (e.g., <code>\$('#a#main').attr('href', 'index.html')</code> ).
removeAttr()	Removes an attribute from an element, e.g., <code>\$('#a#main').removeAttr('href')</code> .
addClass()	Adds a new CSS class to the values in the <code>class</code> attribute. In newer browsers, this is possible without jQuery thanks to the standardized <code>classList</code> property and its <code>add()</code> method.
removeClass()	Removes a CSS class from the values in the <code>class</code> attribute. This too is possible in newer browsers without jQuery thanks to the <code>classList</code> property and its <code>remove()</code> method. However, the <code>removeClass()</code> method can alternatively be passed a function that returns a comma-separated list of CSS classes. This functionality isn't possible via the <code>remove()</code> method of the <code>classList</code> property.
toggleClass()	Toggles a CSS class: if the element has the passed class, the class will be removed, and if the element doesn't have the passed class, the class will be added.

Table 10.16 Methods for Accessing Attributes and CSS Classes

10.2.5 Accessing CSS Properties

To access CSS properties, jQuery provides the `css()` method. Like other methods, such as `html()`, `text()`, and `attr()`, this method can be used to both read and set values. For the former, pass the method the name of the CSS property of which the value is to be read; for the latter, specify the value to be set as the second parameter.

Alternatively, as with the `attr()` method, an object can be passed as an argument, and the respective properties are then used as CSS properties. In addition, you can also pass an array of strings as an argument to read the values of several CSS properties in one step. Some examples are shown in Listing 10.14.

```
// Read the background color of the <body> element
const backgroundColor = $('body').css('background-color');
// Read the foreground color and the background color of the <body> element
```

```
const properties = $('body').css(['color', 'background-color']);
// Set the background color of the <body> element
$('body').css('background-color', 'blue');
// Set the foreground color and the background color of the <body> element
$('body').css({
  'color': 'white',
  'background-color': 'blue'
});
```

Listing 10.14 Accessing CSS Properties

10.2.6 Navigating between Elements

Starting from a selection of elements stored in a jQuery object, you can use the methods presented ahead to find elements that have a specific relationship to these elements, such as parent elements, sibling elements, and child elements. Some examples are shown in Listing 10.15. The corresponding descriptions of the methods can be found in Table 10.17.

```
// Child elements
// Selection of all child elements of <ul>
const listItems = $('ul').children();
// Selection of the next link within <ul>
const closestLink = $('ul').closest('a');

// Sibling elements
// Selection of the next sibling element
const nextSibling = $('ul').next();
// Selection of the next link element
const nextSiblingLink = $('ul').next('a');
// Selection of all next sibling elements
const nextSiblings = $('ul').nextAll();
// Selection of all next link elements
const nextSiblingLinks = $('div').nextAll('a');
// Selection of all next sibling elements up to the specified element
const nextSiblingsUntil = $('div').nextUntil('a');
// Selection of the previous sibling element
const previousSibling = $('ul').prev();
// Selection of all previous sibling elements
const previousSiblings = $('ul').prevAll();
// Selection of all previous sibling elements up to the specified element
const previousSiblingsUntil = $('div').prevUntil('a');
// Selection of all sibling elements
const siblings = $('div').siblings();
```

```
// Parent elements
// Selection of the parent element
const parent = $('ul').parent();
// Selection of all parent elements
const parents = $('ul').parents();
// Selection of all parent elements up to the specified element
const parentsUntil = $('ul').parentsUntil('div');
```

Listing 10.15 Various Examples of Navigating between Elements

Method	Description
children()	Selects the child elements of the selected elements. Optionally, a selector can be passed--in that case only those child elements are selected to which this selector applies..
closest()	Selects the first element of the selected elements that matches the selector passed as a parameter or for which one of the parent elements matches the selector.
next()	Selects the next sibling element of the selected elements. If a selector is passed, the next sibling element that matches this selector is selected.
nextAll()	Selects all next sibling elements of the selected elements. If a selector is passed, the next sibling elements that match this selector are selected.
nextUntil()	Selects all next sibling elements of the selected elements. If a selector is passed, the next sibling elements are selected up to the sibling element that matches this selector.
parent()	Selects the parent element of the selected elements.
parents()	Selects all parent elements preceding in the hierarchy of the selected elements.
parentsUntil()	Selects all parent elements preceding in the hierarchy of the selected elements up to an element that (1) matches the passed selector, (2) matches the passed element, or (3) is contained in the passed jQuery object.
prev()	Selects the previous sibling element of the selected elements. If a selector is passed, the previous sibling element that matches this selector is selected.
prevAll()	Selects all previous sibling elements of the selected elements. If a selector is passed, the previous sibling elements that match this selector are selected.

Table 10.17 Methods for Navigating the DOM Tree

Method	Description
prevUntil()	Selects all previous sibling elements of the selected elements. If a selector is passed, the previous sibling elements are selected up to the sibling element that matches this selector.
siblings()	Selects all sibling elements of the selected elements. If a selector is passed, the sibling elements that match this selector are selected.

Table 10.17 Methods for Navigating the DOM Tree (Cont.)

10.2.7 Using Effects and Animations

Effects such as fading in or out elements of a web page were not always as easy to implement as they are now with the help of CSS3 animations. It's little wonder then that jQuery offers several methods for this as well, the most important of which are shown in Table 10.18. For example, with `fadeIn()`, `fadeOut()`, and `fadeToggle()`, it's possible to fade the selected elements in and out, and `slideDown()`, `slideUp()`, and `slideToggle()` enable you to slide the selected elements in and out.

The most flexible option is provided by `animate()`. This method can be passed—in the form of a configuration object—various CSS properties to be animated, the speed or duration of the animation (either as a string, such as one of the values `fast` or `slow`, or as a numeric value specifying the duration in milliseconds), an *easing function* (which describes how the speed of the animation behaves in relation to the time within the animation), and a callback function that's called when the animation has been fully executed (see Listing 10.16).

Method	Description
animate()	Enables the animation of CSS properties.
clearQueue()	Removes all animations from the queue that have not yet been executed.
delay()	Delays an animation by a specified number of milliseconds.
dequeue()	Executes the next animation in the queue.
fadeIn()	Fades the selected elements in.
fadeOut()	Fades the selected elements out.
fadeTo()	Adjusts the opacity of the selected elements.
fadeToggle()	Fades the selected elements in or out, depending on their state: if an element is visible, it's faded out, but if it isn't visible, it's faded in.

Table 10.18 Methods for Displaying and Hiding Elements

Method	Description
finish()	Stops the current animation, removes all animations from the queue, and sets the CSS properties of the selected elements to the target value.
hide()	Hides the selected elements.
queue()	Accesses the animations in the queue.
show()	Shows the selected elements.
slideDown()	Slides the selected elements down, from top to bottom.
slideToggle()	Slides the selected elements in or out, depending on their state: if an element is visible, it slides out from bottom to the, but if it isn't visible, it slides in from top to bottom.
slideUp()	Slides the selected elements up, from bottom to top.
stop()	Stops the current animation.
toggle()	Hides or displays the selected elements: if an element is visible, it's hidden, but if it isn't visible, it's displayed.

Table 10.18 Methods for Displaying and Hiding Elements (Cont.)

```
'use strict';
$(document).ready(() => {
  $('#main').animate(
    { opacity: 0.75 }, // Properties
    'fast',           // Speed
    'swing',          // Easing
    () => {
      // Animation completed
    }
  );
});
```

Listing 10.16 Accessing CSS Properties

10.3 Responding to Events

As you recall from Chapter 6, there are several options for catching events. Event handlers are usually available for the corresponding event, and there’s also the possibility to register several event listeners for one event via the `addEventListener()` method. Older versions of Internet Explorer also use the `attachEvent()` method, which fulfills a similar task.

We also showed a corresponding browser-independent helper function in Chapter 6. The jQuery library offers a browser-independent solution as well.

10.3.1 Registering Event Listeners

jQuery provides several methods to respond to events or register event listeners. So, on the one hand, you can use the `on()` method, which is called on the jQuery object, as shown in Listing 10.17: you pass the name of the event to be responded to as the first parameter and the event listener in the form of a function as the second parameter.

```
$('#button').on('click', (event) => {
  console.log('Button pressed');
});
```

Listing 10.17 Registering an Event Listener

Note

If the jQuery object contains references to multiple elements, calling an event method for each element in the selection invokes the corresponding event listener.

As an alternative to the general `on()` method, jQuery offers various methods named specifically after the event to be caught, such as the `click()` method found in Listing 10.18. Logically, these event methods don’t need to be passed the name of the respective event as a parameter, but only the event listener.

```
$('#button').click((event) => {
  console.log('Button pressed');
});
```

Listing 10.18 Registering an Event Listener via the Shorthand Method

Overall, the event methods can be classified as follows:

- General event methods (see Table 10.19)
- Event methods for handling general events (Section 10.3.2)
- Event methods for handling mouse events (Section 10.3.3)
- Event methods for handling keyboard events (Section 10.3.4)
- Event methods for handling form events (Section 10.3.5)

Method	Description
bind()	Adds an event listener for an event. Since jQuery 1.7, however, the on() method should be used according to the official documentation.
delegate()	For older jQuery versions, this is the preferred method to add an event listener for an event. However, since jQuery 1.7, the on() method should be used.
off()	Removes an event listener for an event.
on()	Adds an event listener for an event.
one()	Adds an event listener that is triggered at most once per event for each selected element.
trigger()	Runs all event listeners registered for an event.
triggerHandler()	Like the trigger() method, but doesn't perform the default behavior for an event (such as submitting a form).
unbind()	Removes an event listener for an event. Since jQuery 1.7, however, the off() method should be used according to the official documentation.
undelegate()	For older jQuery versions, this is the preferred method to remove an event listener for an event. However, since jQuery 1.7, the off() method should be used.

Table 10.19 Methods for Managing Event Handlers

10.3.2 Responding to General Events

Table 10.20 contains some methods for registering event listeners for general events: `error()` enables you to register event listeners that are triggered when an `error` event is raised on the selected elements, event listeners registered via `ready()` are called as soon as the corresponding elements have been loaded (see Listing 10.19), event listeners registered via `resize()` are called whenever a `resize` event occurs for the elements, and event listeners registered via `scroll()` are called whenever a `scroll` event occurs for the elements.

Method	Description
error()	Register event listeners that are executed when an <code>error</code> event occurs.
ready()	Register event listeners that are executed when the DOM or the element passed to the method is fully loaded.

Table 10.20 Different Methods for Registering Event Listeners

Method	Description
resize()	Register event listeners that are executed when a <code>resize</code> event occurs.
scroll()	Register event listeners that are executed when an element is scrolled.

Table 10.20 Different Methods for Registering Event Listeners (Cont.)

```
$(document).ready(() => {
  console.log('Web page loaded');
});
```

Listing 10.19 Registering an Event Listener for Loading the Document

10.3.3 Responding to Mouse Events

Table 10.21 shows the methods jQuery uses to register event listeners for mouse events. They basically correspond to the events you already know from Chapter 6: `click()` and `dblclick()` for mouse clicks; `focusin()` and `focusout()` for focusing elements; and `mousedown()`, `mouseenter()`, `mouseleave()`, `mousemove()`, `mouseout()`, `mouseover()`, and `mouseup()` for mouse movements over elements.

Method	Description
click()	Register event listeners that are executed when the mouse is clicked
dblclick()	Register event listeners that are executed when the mouse is double-clicked
focusin()	Register event listeners that are executed when an element receives focus
focusout()	Register event listeners that are executed when an element loses focus
hover()	Register event listeners that are executed when the mouse pointer hovers over an element
mousedown()	Register event listeners that are executed when the mouse pointer is over an element and the mouse button is pressed
mouseenter()	Register event listeners that are executed when the mouse pointer enters an element
mouseleave()	Register event listeners that are executed when the mouse pointer leaves an element
mousemove()	Register event listeners that are triggered when the mouse moves over an element

Table 10.21 Methods for Handling Mouse Events

Method	Description
<code>mouseout()</code>	Register event listeners that are executed when the mouse pointer leaves an element
<code>mouseover()</code>	Register event listeners that are executed when the mouse pointer enters an element
<code>mouseup()</code>	Register event listeners that are executed when the mouse pointer is over an element and the mouse button is released

Table 10.21 Methods for Handling Mouse Events (Cont.)

An example is shown in Listing 10.20.

```
$('#button#target').click((event) => {
  console.log('Button was pressed');
});
```

Listing 10.20 Registering an Event Listener for a Mouse Event

By the way, it’s also possible to use the event methods not for registering event listeners, but for triggering events. To do this, simply call the corresponding method without any arguments. In Listing 10.21, for example, within the second event listener (registered on the `<button>` element with the ID `target2`), the `click` event is triggered for the `<button>` element with the ID `target`.

```
$('#button#target').click((event) => {
  console.log('Button was pressed');
});
$('#button#target2').click((event) => {
  $('#button#target').click();
});
```

Listing 10.21 Triggering an Event

Note

Most methods in jQuery can be called with a different number of arguments, and each has different functions. For example, as shown earlier in this chapter, you can use the `attr()` method to both read and write HTML attributes or, as just shown, you can use the event methods to both register event listeners and trigger events.

10.3.4 Responding to Keyboard Events

For registering event listeners for keyboard events, the methods listed in Table 10.22 are available: `keydown()`, `keyup()`, and `keypress()` for registering event listeners that are triggered when a key is pressed or released. Again, all of this should be familiar from Chapter 6.

Method	Description
<code>keydown()</code>	Register event listeners that are executed when a key on the keyboard is pressed. If a key is pressed for a longer time, the event listener is executed several times.
<code>keypress()</code>	Register event listeners that are executed when a key on the keyboard is pressed.
<code>keyup()</code>	Register event listeners that are executed when a key on the keyboard is released.

Table 10.22 Methods for Handling Keyboard Events

An example of using these methods is shown in Listing 10.22. This nicely illustrates how the individual event methods (or all jQuery methods in general) can be used one after the other.

```
$('#input#username')
  .keypress((event) => {
    console.log('Key for entering username pressed.');
```

```
  })
  .keydown((event) => {
    console.log('Key is pressed.');
```

```
  })
  .keyup((event) => {
    console.log('Key for entering username released.');
```

```
  });
```

Listing 10.22 Registering Different Event Listeners for Keyboard Events

Fluent API

When an API allows you to call a method directly on the return value of a method, as in Listing 10.22, it is also called a fluent API.

10.3.5 Responding to Form Events

The methods listed in Table 10.23 for registering event listeners related to form events should also be essentially familiar from Chapter 6: `blur()` and `focus()` for registering



event listeners that are triggered when a form field loses or receives focus, `change()` when the value of a form field changes, `select()` when a specific value is selected for a form field, and `submit()` when a form is submitted.

Method	Description
<code>blur()</code>	Register event listeners that are executed when a form field loses focus
<code>change()</code>	Register event listeners that are executed when the selected value of a selection list, a checkbox, or a group of radio buttons has been changed
<code>focus()</code>	Register event listeners that are executed when a form field receives focus
<code>select()</code>	Register event listeners that are executed when the text of an input field (<input> element of the text type) or a text input area (<textarea> element) is selected
<code>submit()</code>	Register event listeners that are executed when a form is submitted

Table 10.23 Methods for Handling Form Events

Some examples are shown in Listing 10.23.

```
$('#input#username')
  .focus((event) => {
    console.log('Input field focused.');
```

```
  })
  .blur((event) => {
    console.log('Input field no longer focused.');
```

```
  })
  .change((event) => {
    console.log('Text changed.');
```

```
  })
  .select((event) => {
    console.log('Text selected.');
```

```
  });
```

Listing 10.23 Registering Different Event Listeners for Form Events

10.3.6 Accessing Information from Events

The *event object*, which is available as a parameter within each event listener, contains different information and provides different methods, as shown in Table 10.24. Basically, this is the information also contained in the standard event object, as discussed in Chapter 6, supplemented by a few more details. But again, jQuery hides the browser-specific details, allowing for browser-independent use.

Property/Method	Description
<code>currentTarget</code>	Contains the current element during the bubbling phase
<code>data</code>	Contains an optional data object passed to the event method
<code>delegateTarget</code>	Contains the element on which the event listener was registered
<code>isDefaultPrevented()</code>	Indicates whether <code>preventDefault()</code> was called on the event object
<code>isImmediatePropagationStopped()</code>	Indicates whether <code>stopImmediatePropagation()</code> was called on the event object
<code>isPropagationStopped()</code>	Indicates whether <code>stopPropagation()</code> was called on the event object
<code>metaKey</code>	Contains an indication of whether the so-called meta key (for Mac keyboards, the <code>cmd</code> key; for Windows keyboards, the <code>Windows</code> key) was pressed while the event was triggered.
<code>namespace</code>	Contains the namespace of the event
<code>pageX</code>	Contains the mouse position relative to the left edge of the document
<code>pageY</code>	Contains the mouse position relative to the top of the document
<code>preventDefault()</code>	Prevents the default action for an event from being executed
<code>relatedTarget</code>	For an element for which an event was triggered, contains the element directly related to the event (e.g., in the case of a <code>mouseout</code> event, the element on which the <code>mouseover</code> event was triggered by the same user action)
<code>result</code>	Contains the result of an event listener previously triggered for an event
<code>stopImmediatePropagation()</code>	Immediately prevents the event from rising further during the bubbling phase
<code>stopPropagation()</code>	Prevents the event from rising further during the bubbling phase

Table 10.24 Properties and Methods of the Event Object

Property/Method	Description
target	Contains the element that triggered the event
timeStamp	Contains a timestamp indicating the time when the event was triggered
type	Contains the type of the event
which	In the case of mouse or keyboard events, contains the mouse button or key on the keyboard that was pressed

Table 10.24 Properties and Methods of the Event Object (Cont.)

Listing 10.24 shows an example of how you can access this information. Most importantly, you can also see here that it’s possible to pass an event a data object that you can then access within the event listener.

```
$('#input').on(
  'change',
  {
    value : 4711           // Data object
  },
  (event) => {
    console.log(event.currentTarget); // current element
    console.log(event.data);         // data object
    console.log(event.data.value);   // property of the data object
    console.log(event.pageX);        // x position of mouse
    console.log(event.pageY);        // y position of mouse
  }
);
```

Listing 10.24 Accessing the jQuery Event Object

10.4 Creating Ajax Requests

Generating Ajax requests is also considerably simplified by jQuery. In this section, we’ll show you how to perform the examples in Chapter 9 for Ajax-based loading of HTML, XML, and JSON data via the appropriate jQuery methods.

10.4.1 Creating Ajax Requests

For creating Ajax requests, jQuery provides several methods, listed in Table 10.25. These are methods that—with the exception of the `load()` method—are called not on a

selection of elements, as has been the case so far in this chapter, but directly on the `$` object. Therefore, we call these methods *global jQuery methods* ahead.

Method	Description
<code>\$.ajax()</code>	Performs an asynchronous HTTP request
<code>\$.get()</code>	Performs an HTTP request using the HTTP GET method
<code>\$.getJSON()</code>	Performs an HTTP GET request to load JSON data from a server
<code>\$.getScript()</code>	Performs an HTTP GET request to load JavaScript data from a server and execute it directly
<code>load()</code>	Performs an HTTP GET request to load HTML data from a server and embed it directly into the selected elements
<code>\$.post()</code>	Performs an HTTP request using the HTTP POST method

Table 10.25 Main Methods for Working with Ajax

The jQuery global method `ajax()` (or `$.ajax()`) allows you to create arbitrary Ajax requests. The configuration object expected by this method gives you the most leeway regarding the configuration of a request.

The `get()` and `post()` methods are used to create GET or POST requests, meaning that you don’t have to worry about specific configurations for these request types, such as specifying the HTTP method.

In addition, special methods are available for loading HTML data (`load()`), loading JSON data (`getJSON()`), and loading JavaScript files (`getScript()`).

Listing 10.25 shows an example of using the `ajax()` method, which you already know about in principle from Chapter 9: the goal is to load JSON data from the server and dynamically create a table for this data.

The URL for the request is configured via the `url` property of the configuration object, the type expected as a response via the `dataType` property (possibilities include, for example, `json`, `xml`, or `html`), and the type of the request via the `type` property. Callback functions for the successful execution of a request or also for errors can be defined via the `success` and `error` properties. Within the `success` callback function, the response of the server is accessed via the `data` parameter in the example. Because this is JSON data, it can be processed directly to create the table, as already mentioned.

```
'use strict';
$(document).ready(() => {
  $.ajax({
    url: 'artists.json',
    dataType: 'json',
    type: 'GET',
```

```

    success: (data) => {
      const table = initTable();
      const artists = data.artists;
      for (let i = 0; i < artists.length; i++) {
        const artist = artists[i];
        const albums = artist.albums;
        for (let j = 0; j < albums.length; j++) {
          const album = albums[j];
          const row = createRow(
            artist.name,
            album.title,
            album.year
          );
          $(table).find('tbody').append(row);
        }
      }
      $('#artists-container').append(table);
    },
    error: (jqXHR, errorMessage, error) => {
    }
  });
});
```

Listing 10.25 Generating an Ajax Request

As an alternative to specifying the callback functions via the `success` and `error` properties, you also have the option of defining them via the `done()` and `fail()` methods, which (thanks to jQuery's Fluent API) can be combined directly with calling the `ajax()` method (see Listing 10.26).

```

'use strict';
$(document).ready(() => {
  $.ajax({
    url: 'artists.json',
    dataType: 'json',
    type: 'GET'
  })
  .done((data) => {
    const table = initTable();
    const artists = data.artists;
    for (let i = 0; i < artists.length; i++) {
      const artist = artists[i];
      const albums = artist.albums;
      for (let j = 0; j < albums.length; j++) {
        const album = albums[j];
```

```

    const row = createRow(
      artist.name,
      album.title,
      album.year
    );
    $(table).find('tbody').append(row);
  }
}
$('#artists-container').append(table);
})
.fail((jqXHR, errorMessage, error) => {
});
});
```

Listing 10.26 Generating an Ajax Request via the Fluent API

10.4.2 Responding to Events

For responding to events related to working with Ajax requests, jQuery provides the methods shown in Table 10.26.

Method	Description
<code>ajaxComplete()</code>	Specify an event listener that is called when an Ajax request completes
<code>ajaxError()</code>	Specify an event listener for errors
<code>ajaxSend()</code>	Specify an event listener that is called when an Ajax request is sent
<code>ajaxStart()</code>	Specify an event listener that is called when the first Ajax request is started
<code>ajaxStop()</code>	Specify an event listener that is called when all Ajax requests have completed
<code>ajaxSuccess()</code>	Specify an event listener that is called whenever an Ajax request completes successfully

Table 10.26 Methods for Handling Ajax Events

An example is shown in Listing 10.27. There are two things to keep in mind here: first, the methods are each called on a selection of elements (or the corresponding jQuery object); second, the event listeners each have a different number of parameters. The event listeners for `ajaxStart()` and `ajaxStop()` have no parameters at all, the event listeners for `ajaxSend()` and `ajaxComplete()` each get the event object, plus an object representing the Ajax request and an object with configurations related to the request. The event listeners for the `ajaxSuccess()` and `ajaxError()` methods also receive the response data and the error object, respectively.

```
$(document)
  .ajaxStart(() => {
    console.log('Request started.');
```

```
  })
  .ajaxSend((event, request, settings) => {
    console.log('Request sent.');
```

```
  })
  .ajaxSuccess((event, request, settings, data) => {
    console.log('Request completed successfully');
```

```
  })
  .ajaxError((event, request, settings, error) => {
    console.log('Error on request: ' + error);
```

```
  })
  .ajaxComplete((event, request, settings) => {
    console.log('Request completed.');
```

```
  })
  .ajaxStop(() => {
    console.log('All requests completed.');
```

```
  });
```

**Listing 10.27** Registering Different Event Listeners for Ajax Events

### 10.4.3 Loading HTML Data via Ajax

To load HTML data via Ajax, you can proceed as in Listing 10.28 and use the global jQuery `get()` method. The important thing here is that you pass the value `html` to the `dataType` property. You can then use `html()` in the corresponding callback function to assign the response data directly to an element as HTML content.

```
'use strict';
$(document).ready(() => {
  const login = $('#login');
  const register = $('#register');
  login.click((e) => {
    e.preventDefault();
    loadContent('login');
```

```
  });
  register.click((e) => {
    e.preventDefault();
    loadContent('register');
```

```
  });
});

function loadContent(name) {
  $.get({
```

```
    url: name + '.html',
    dataType: 'html'
  }).done((data) => {
    $('#main-content').html(data);
  });
}
```

**Listing 10.28** Loading HTML Data via Ajax

But this process is even easier with the `load()` method, as shown in Listing 10.29. You can call this method directly on a jQuery object (or the selection of elements it represents). As arguments, you pass the URL from which the HTML data should be loaded and optionally a callback function that will be called when the data has been successfully loaded.

```
function loadContent(name) {
  $('#main-content').load(
    name + '.html',
    (
      responseText,
      textStatus,
      jqXHRObject
    ) => {
      console.log('HTML loaded');
```

```
    }
  );
}
```

**Listing 10.29** Alternative Loading of HTML Data via Ajax

#### Sending Additional Data with the Request

You can optionally insert another argument between the URL and the callback function—namely to define the data to be sent to the server with the request (in the form of a string). This is useful, for example, if the server is to generate either this or that response based on the data.

### 10.4.4 Loading XML Data via Ajax

To load XML data, use the `get()` method as shown in Listing 10.30, passing the `xml` value for the `dataType` property. Within the callback function, the response data is then directly available as XML or as a DOM tree. The best thing about this is that the jQuery `$()` method can also use it—for example, to select all `<artist>` elements using the `find()` method as shown in the listing, to iterate over these elements using `each()`

(another helper method of jQuery, by the way), or to access the text content of the `<title>` and `<year>` elements using `text()`.

```
'use strict';
$(document).ready(() => {
  $.get({
    url: 'artists.xml',
    dataType: 'xml'
  }).done((data) => {
    const table = initTable();
    const artists = $(data).find('artist');
    artists.each((index, artist) => {
      const albums = $(artist).find('album');
      albums.each((index, album) => {
        const row = createRow(
          artist.getAttribute('name'),
          $(album).find('title').text(),
          $(album).find('year').text()
        );
        $(table).find('tbody').append(row);
      });
    });
    $('#artists-container').append(table);
  });
});
```

**Listing 10.30** Loading XML Data via Ajax

#### 10.4.5 Loading JSON Data via Ajax

We showed you how to load JSON data using jQuery at the beginning of Section 10.4.1 using the `ajax()` method. Listing 10.31 shows the equivalent example using the `get()` method. You specify `json` as the value for the `dataType` property and can then access the JSON data sent by the server in the callback function as usual.

```
'use strict';
$(document).ready(() => {
  $.get({
    url: 'artists.json',
    dataType: 'json'
  }).done((data) => {
    const table = initTable();
    const artists = data.artists;
    for (let i = 0; i < artists.length; i++) {
      const artist = artists[i];
```

```
const albums = artist.albums;
for (let j = 0; j < albums.length; j++) {
  const album = albums[j];
  const row = createRow(
    artist.name,
    album.title,
    album.year
  );
  $(table).find('tbody').append(row);
}
}
$('#artists-container').append(table);
});
});
```

**Listing 10.31** Loading JSON Data via Ajax

Alternatively, jQuery provides the `getJSON()` method, which further simplifies requesting JSON data (see Listing 10.32). As arguments, you pass this method the URL to be requested and a callback function to access the JSON data sent by the server.

```
'use strict';
$(document).ready(() => {
  $.getJSON(
    'artists.json',
    (
      data,
      textStatus,
      jqXHRObject
    ) => {
      // here is the already known content
    }
  );
});
```

**Listing 10.32** Alternative Loading of JSON Data via Ajax

#### Sending Additional Data with the Request

As you saw with the `load()` method, you can optionally specify one more argument between the URL and the callback function to define those data that should be sent to the server with the request.



10.5 Summary

In this chapter, you learned about the popular jQuery JavaScript library, which simplifies many things, especially with regard to DOM manipulation, event handling, and creating Ajax requests. The following list summarizes the most important aspects:

- jQuery is a library that mainly hides browser-specific details and provides helper methods for recurring tasks that can be used across browsers.
- The linchpin for working with jQuery is the `jQuery()` or `$()` method.
- Among other things, you can pass a selector, an existing element, or an HTML string as an argument to this method.
- As a return value, the method provides a wrapper object (*jQuery object*) that extends the corresponding elements by additional methods (*jQuery methods*).
- Thus, a jQuery object provides various methods for working with the DOM, including the following:
  - Methods to access the content of elements
  - Methods to filter selected elements
  - Methods to access attributes
  - Methods to access CSS properties
  - Methods to navigate between elements
  - Methods to animate elements or their CSS properties
- For working with events, jQuery provides several methods to register event listeners, including the following:
  - Methods to register event listeners for general events
  - Methods to register event listeners for mouse events
  - Methods to register event listeners for keyboard events
  - Methods to register event listeners for form events
- Creating Ajax requests is also made easier by a number of helper methods, including the following:
  - A method to create arbitrary Ajax requests
  - A method to create GET requests
  - A method to create POST requests
  - A method to load HTML content directly into an element via Ajax
  - A method to load JavaScript files
  - A method to load JSON files
- Most helper methods can be used for various purposes; for example, HTML attributes can be both read and written via the `attr()` method, CSS properties can be read and written via the `css()` method, and event listeners can be registered or removed again via the event methods.

Finally, Table 10.27 compares how different problems can be handled with jQuery and with pure JavaScript. For more examples, we recommend looking at the <http://you-mightnotneedjquery.com> website mentioned earlier. There you can see very nicely how the code of both variants is about equally compact, especially in the DOM manipulation area. When working with events and with Ajax, the code is still a bit more compact with jQuery. In all cases, however, it's true that jQuery is largely browser-independent, while this doesn't always apply to the pure JavaScript variants.

Working with the DOM	
Add CSS class	jQuery: <pre>\$(element).addClass(   newClassName );</pre>
	Pure JavaScript: <pre>if (element.classList) {   element.classList.add(newClassName); } else {   element.className += ' ' + newClassName; }</pre>
Access child elements	jQuery: <pre>\$(element).children();</pre>
	Pure JavaScript: <pre>element.children</pre>
Iterate over elements	jQuery: <pre>\$(selector).each(   (index, element) =&gt; {   } );</pre>
	Pure JavaScript: <pre>const elements = document.querySelectorAll(   selector ); Array.prototype.forEach.call(   elements, (element, index) =&gt; {   } );</pre>

Table 10.27 Comparison between jQuery and Pure JavaScript

Working with the DOM	
Search elements below an element	jQuery: \$(element).find(selector);
	Pure JavaScript: element.querySelectorAll(selector);
Search elements	jQuery: \$(selector);
	Pure JavaScript: document.querySelectorAll(selector);
Access attributes	jQuery: \$(element).attr(name);
	Pure JavaScript: element.getAttribute(name);
Read HTML content	jQuery: \$(element).html();
	Pure JavaScript: element.innerHTML;
Write HTML content	jQuery: \$(element).html(content);
	Pure JavaScript: element.innerHTML = content;
Read text content	jQuery: \$(element).text();
	Pure JavaScript: element.textContent;

Table 10.27 Comparison between jQuery and Pure JavaScript (Cont.)

Working with the DOM	
Write text content	jQuery: \$(element).text(content);
	Pure JavaScript: element.textContent = content;
Next element	jQuery: \$(element).next();
	Pure JavaScript: element.nextElementSibling;
Previous element	jQuery: \$(element).prev();
	Pure JavaScript: element.previousElementSibling;
Working with events	
Add event listener	jQuery: \$(element).on(eventName, eventHandler);
	Pure JavaScript: element.addEventListener(eventName, eventHandler);
Remove event listener	jQuery: \$(element).off(eventName, eventHandler);
	Pure JavaScript: element.removeEventListener(eventName, eventHandler);

Table 10.27 Comparison between jQuery and Pure JavaScript (Cont.)

Working with the DOM	
Execute function when loading the document	jQuery: \$(document).ready(() => { });
	Pure JavaScript: function ready(callback) { if (document.readyState != 'loading'){ callback(); } else { document.addEventListener( 'DOMContentLoaded', callback ); } }
Working with Ajax requests	
Send GET request	jQuery: \$.ajax({ type: 'GET', url: url, success: response => {}, error: () => {} });
	Pure JavaScript: fetch(url) .then(response => {}) .catch(error => {});
Send POST request	jQuery: \$.ajax({ type: 'POST', url: url, data: data });
	Pure JavaScript: fetch('url', { method: 'POST', body: data, }) .then(response => {}) .catch(error => {});

Table 10.27 Comparison between jQuery and Pure JavaScript (Cont.)

Working with the DOM	
Load JSON via Ajax	jQuery: \$.getJSON( 'data.json', (data) => {} );
	Pure JavaScript: fetch('data.json') .then(response => response.json()) .then(data => {}) .catch(error => {});

Table 10.27 Comparison between jQuery and Pure JavaScript (Cont.)

# Contents

Book Resources .....	25
Preface .....	27
<b>1 Basics and Introduction</b> .....	<b>31</b>
<b>1.1 Programming Basics</b> .....	<b>31</b>
1.1.1 Communicating with the Computer .....	32
1.1.2 Programming Languages .....	33
1.1.3 Tools for Program Design .....	40
<b>1.2 Introduction to JavaScript</b> .....	<b>46</b>
1.2.1 History .....	46
1.2.2 Fields of Application .....	47
<b>1.3 Summary</b> .....	<b>53</b>
<b>2 Getting Started</b> .....	<b>55</b>
<b>2.1 Introduction to JavaScript and Web Development</b> .....	<b>55</b>
2.1.1 The Relationship among HTML, CSS, and JavaScript .....	55
2.1.2 The Right Tool for Development .....	59
<b>2.2 Integrating JavaScript into a Web Page</b> .....	<b>64</b>
2.2.1 Preparing a Suitable Folder Structure .....	64
2.2.2 Creating a JavaScript File .....	65
2.2.3 Embedding a JavaScript File in an HTML File .....	66
2.2.4 Defining JavaScript Directly within the HTML .....	69
2.2.5 Placement and Execution of the <script> Elements .....	70
2.2.6 Displaying the Source Code .....	74
<b>2.3 Creating Output</b> .....	<b>76</b>
2.3.1 Showing the Standard Dialog Window .....	76
2.3.2 Writing to the Console .....	78
2.3.3 Using Existing UI Components .....	81
<b>2.4 Summary</b> .....	<b>83</b>

<b>3</b>	<b>Language Core</b>	85
<b>3.1</b>	<b>Storing Values in Variables</b>	85
3.1.1	Defining Variables	85
3.1.2	Using Valid Variable Names	88
3.1.3	Defining Constants	94
<b>3.2</b>	<b>Using the Different Data Types</b>	94
3.2.1	Numbers	95
3.2.2	Strings	98
3.2.3	Boolean Values	103
3.2.4	Arrays	104
3.2.5	Objects	109
3.2.6	Special Data Types	110
3.2.7	Symbols	112
<b>3.3</b>	<b>Deploying the Different Operators</b>	112
3.3.1	Operators for Working with Numbers	114
3.3.2	Operators for Easier Assignment	115
3.3.3	Operators for Working with Strings	116
3.3.4	Operators for Working with Boolean Values	117
3.3.5	Operators for Working with Bits	124
3.3.6	Operators for Comparing Values	125
3.3.7	The Optional Chaining Operator	128
3.3.8	The Logical Assignment Operators	130
3.3.9	Operators for Special Operations	132
<b>3.4</b>	<b>Controlling the Flow of a Program</b>	132
3.4.1	Defining Conditional Statements	133
3.4.2	Defining Branches	134
3.4.3	Using the Selection Operator	140
3.4.4	Defining Multiway Branches	142
3.4.5	Defining Counting Loops	148
3.4.6	Defining Head-Controlled Loops	155
3.4.7	Defining Tail-Controlled Loops	158
3.4.8	Prematurely Terminating Loops and Loop Iterations	160
<b>3.5</b>	<b>Creating Reusable Code Blocks</b>	168
3.5.1	Defining Functions	168
3.5.2	Calling Functions	171
3.5.3	Passing and Evaluating Function Parameters	172
3.5.4	Defining Return Values	180
3.5.5	Defining Default Values for Parameters	182
3.5.6	Using Elements from an Array as Parameters	184
3.5.7	Defining Functions Using Short Notation	185

3.5.8	Modifying Strings via Functions	188
3.5.9	Functions in Detail	189
3.5.10	Calling Functions through User Interaction	197
<b>3.6</b>	<b>Responding to Errors and Handling Them Correctly</b>	198
3.6.1	Syntax Errors	198
3.6.2	Runtime Errors	199
3.6.3	Logic Errors	200
3.6.4	The Principle of Error Handling	201
3.6.5	Catching and Handling Errors	202
3.6.6	Triggering Errors	205
3.6.7	Errors and the Function Call Stack	208
3.6.8	Calling Certain Statements Regardless of Errors That Have Occurred	210
<b>3.7</b>	<b>Commenting the Source Code</b>	216
<b>3.8</b>	<b>Debugging the Code</b>	216
3.8.1	Introduction	217
3.8.2	A Simple Code Example	217
3.8.3	Defining Breakpoints	218
3.8.4	Viewing Variable Assignments	220
3.8.5	Running a Program Step by Step	221
3.8.6	Defining Multiple Breakpoints	223
3.8.7	Other Types of Breakpoints	223
3.8.8	Viewing the Function Call Stack	224
<b>3.9</b>	<b>Summary</b>	226
<b>4</b>	<b>Working with Reference Types</b>	229
<b>4.1</b>	<b>Difference between Primitive Data Types and Reference Types</b>	229
4.1.1	The Principle of Primitive Data Types	229
4.1.2	The Principle of Reference Types	230
4.1.3	Primitive Data Types and Reference Types as Function Arguments	232
4.1.4	Determining the Type of a Variable	233
4.1.5	Outlook	236
<b>4.2</b>	<b>Encapsulating State and Behavior in Objects</b>	236
4.2.1	Introduction to Object-Oriented Programming	236
4.2.2	Creating Objects Using Literal Notation	237
4.2.3	Creating Objects via Constructor Functions	239
4.2.4	Creating Objects Using Classes	242
4.2.5	Creating Objects via the Object.create() Function	246



4.2.6	Accessing Properties and Calling Methods .....	249
4.2.7	Adding or Overwriting Object Properties and Object Methods .....	256
4.2.8	Deleting Object Properties and Object Methods .....	260
4.2.9	Outputting Object Properties and Object Methods .....	262
4.2.10	Using Symbols to Define Unique Object Properties .....	265
4.2.11	Preventing Changes to Objects .....	267
<b>4.3</b>	<b>Working with Arrays .....</b>	<b>270</b>
4.3.1	Creating and Initializing Arrays .....	270
4.3.2	Accessing Elements of an Array .....	273
4.3.3	Adding Elements to an Array .....	274
4.3.4	Removing Elements from an Array .....	279
4.3.5	Copying Some of the Elements from an Array .....	282
4.3.6	Sorting Arrays .....	284
4.3.7	Using Arrays as a Stack .....	287
4.3.8	Using Arrays as a Queue .....	288
4.3.9	Finding Elements in Arrays .....	290
4.3.10	Copying Elements within an Array .....	292
4.3.11	Converting Arrays to Strings .....	293
<b>4.4</b>	<b>Extracting Values from Arrays and Objects .....</b>	<b>294</b>
4.4.1	Extracting Values from Arrays .....	294
4.4.2	Extracting Values from Objects .....	298
4.4.3	Extracting Values within a Loop .....	302
4.4.4	Extracting Arguments of a Function .....	303
4.4.5	Copying Object Properties to Another Object .....	304
4.4.6	Copying Object Properties from Another Object .....	305
<b>4.5</b>	<b>Working with Strings .....</b>	<b>306</b>
4.5.1	The Structure of a String .....	306
4.5.2	Determining the Length of a String .....	307
4.5.3	Searching within a String .....	308
4.5.4	Extracting Parts of a String .....	310
<b>4.6</b>	<b>Using Maps .....</b>	<b>314</b>
4.6.1	Creating Maps .....	314
4.6.2	Basic Operations .....	315
4.6.3	Iterating over Maps .....	317
4.6.4	Using Weak Maps .....	319
<b>4.7</b>	<b>Using Sets .....</b>	<b>321</b>
4.7.1	Creating Sets .....	321
4.7.2	Basic Operations of Sets .....	321
4.7.3	Iterating over Sets .....	323
4.7.4	Using Weak Sets .....	324

<b>4.8</b>	<b>Other Global Objects .....</b>	<b>325</b>
4.8.1	Working with Date and Time Information .....	325
4.8.2	Performing Complex Calculations .....	328
4.8.3	Wrapper Objects for Primitive Data Types .....	329
<b>4.9</b>	<b>Working with Regular Expressions .....</b>	<b>329</b>
4.9.1	Defining Regular Expressions .....	330
4.9.2	Testing Characters against a Regular Expression .....	330
4.9.3	Using Character Classes .....	333
4.9.4	Limiting Beginning and End .....	336
4.9.5	Using Quantifiers .....	339
4.9.6	Searching for Occurrences .....	343
4.9.7	Searching All Occurrences within a String .....	344
4.9.8	Accessing Individual Parts of an Occurrence .....	345
4.9.9	Searching for Specific Strings .....	346
4.9.10	Replacing Occurrences within a String .....	347
4.9.11	Searching for Occurrences .....	347
4.9.12	Splitting Strings .....	348
<b>4.10</b>	<b>Functions as Reference Types .....</b>	<b>349</b>
4.10.1	Using Functions as Arguments .....	349
4.10.2	Using Functions as Return Values .....	351
4.10.3	Standard Methods of Each Function .....	353
<b>4.11</b>	<b>Summary .....</b>	<b>356</b>
<b>5</b>	<b>Dynamically Changing Web Pages .....</b>	<b>357</b>
<b>5.1</b>	<b>Structure of a Web Page .....</b>	<b>357</b>
5.1.1	Document Object Model .....	357
5.1.2	The Different Types of Nodes .....	358
5.1.3	The Document Node .....	361
<b>5.2</b>	<b>Selecting Elements .....</b>	<b>363</b>
5.2.1	Selecting Elements by ID .....	364
5.2.2	Selecting Elements by Class .....	367
5.2.3	Selecting Elements by Element Name .....	370
5.2.4	Selecting Elements by Name .....	371
5.2.5	Selecting Elements by Selector .....	373
5.2.6	Selecting the Parent Element of an Element .....	378
5.2.7	Selecting the Child Elements of an Element .....	381
5.2.8	Selecting the Sibling Elements of an Element .....	385
5.2.9	Calling Selection Methods on Elements .....	387

5.2.10	Selecting Elements by Type .....	389
<b>5.3</b>	<b>Working with Text Nodes .....</b>	<b>390</b>
5.3.1	Accessing the Text Content of an Element .....	391
5.3.2	Modifying the Text Content of an Element .....	391
5.3.3	Modifying the HTML below an Element .....	392
5.3.4	Creating and Adding Text Nodes .....	393
<b>5.4</b>	<b>Working with Elements .....</b>	<b>394</b>
5.4.1	Creating and Adding Elements .....	394
5.4.2	Removing Elements and Nodes .....	397
5.4.3	The Different Types of HTML Elements .....	398
<b>5.5</b>	<b>Working with Attributes .....</b>	<b>403</b>
5.5.1	Reading the Value of an Attribute .....	403
5.5.2	Changing the Value of an Attribute or Adding a New Attribute .....	405
5.5.3	Creating and Adding Attribute Nodes .....	406
5.5.4	Removing Attributes .....	406
5.5.5	Accessing CSS classes .....	406
<b>5.6</b>	<b>Summary .....</b>	<b>408</b>
<b>6</b>	<b>Processing and Triggering Events .....</b>	<b>409</b>
<b>6.1</b>	<b>The Concept of Event-Driven Programming .....</b>	<b>409</b>
<b>6.2</b>	<b>Responding to Events .....</b>	<b>410</b>
6.2.1	Defining an Event Handler via HTML .....	412
6.2.2	Defining an Event Handler via JavaScript .....	415
6.2.3	Defining Event Listeners .....	417
6.2.4	Defining Multiple Event Listeners .....	418
6.2.5	Passing Arguments to Event Listeners .....	420
6.2.6	Removing Event Listeners .....	422
6.2.7	Defining Event Handlers and Event Listeners via a Helper Function ....	423
6.2.8	Accessing Information of an Event .....	424
<b>6.3</b>	<b>The Different Types of Events .....</b>	<b>426</b>
6.3.1	Events when Interacting with the Mouse .....	427
6.3.2	Events when Interacting with the Keyboard and with Text Fields .....	431
6.3.3	Events when Working with Forms .....	434
6.3.4	Events when Focusing Elements .....	435
6.3.5	General Events of the User Interface .....	435
6.3.6	Events on Mobile Devices .....	438

<b>6.4</b>	<b>Understanding and Influencing the Flow of Events .....</b>	<b>439</b>
6.4.1	The Event Phases .....	439
6.4.2	Interrupting the Event Flow .....	447
6.4.3	Preventing Default Actions of Events .....	452
<b>6.5</b>	<b>Programmatically Triggering Events .....</b>	<b>454</b>
6.5.1	Triggering Simple Events .....	454
6.5.2	Triggering Events with Passed Arguments .....	455
6.5.3	Triggering Default Events .....	456
<b>6.6</b>	<b>Summary .....</b>	<b>456</b>
<b>7</b>	<b>Working with Forms .....</b>	<b>459</b>
<b>7.1</b>	<b>Accessing Forms and Form Fields .....</b>	<b>459</b>
7.1.1	Accessing Forms .....	459
7.1.2	Accessing Form Elements .....	463
7.1.3	Reading the Value of Text Fields and Password Fields .....	465
7.1.4	Reading the Value of Checkboxes .....	467
7.1.5	Reading the Value of Radio Buttons .....	467
7.1.6	Reading the Value of Selection Lists .....	469
7.1.7	Reading the Values of Multiple Selection Lists .....	470
7.1.8	Populating Selection Lists with Values Using JavaScript .....	471
<b>7.2</b>	<b>Programmatically Submitting and Resetting Forms .....</b>	<b>472</b>
<b>7.3</b>	<b>Validating Form Inputs .....</b>	<b>475</b>
<b>7.4</b>	<b>Summary .....</b>	<b>485</b>
<b>8</b>	<b>Controlling Browsers and Reading Browser Information .....</b>	<b>487</b>
<b>8.1</b>	<b>The Browser Object Model .....</b>	<b>487</b>
<b>8.2</b>	<b>Accessing Window Information .....</b>	<b>489</b>
8.2.1	Determining the Size and Position of a Browser Window .....	489
8.2.2	Changing the Size and Position of a Browser Window .....	490
8.2.3	Accessing Display Information of the Browser Bars .....	492
8.2.4	Determining General Properties .....	493
8.2.5	Opening New Browser Windows .....	494
8.2.6	Closing the Browser Window .....	495

8.2.7	Opening Dialogs .....	496
8.2.8	Executing Functions in a Time-Controlled Manner .....	497
<b>8.3</b>	<b>Accessing Navigation Information of a Currently Open Web Page .....</b>	<b>499</b>
8.3.1	Accessing the Individual Components of the URL .....	499
8.3.2	Accessing Query String Parameters .....	500
8.3.3	Loading a New Web Page .....	500
<b>8.4</b>	<b>Viewing and Modifying the Browsing History .....</b>	<b>502</b>
8.4.1	Navigating in the Browsing History .....	502
8.4.2	Browsing History for Single-Page Applications .....	503
8.4.3	Adding Entries to the Browsing History .....	503
8.4.4	Responding to Changes in the Browsing History .....	506
8.4.5	Replacing the Current Entry in the Browsing History .....	506
<b>8.5</b>	<b>Recognizing Browsers and Determining Browser Features .....</b>	<b>508</b>
<b>8.6</b>	<b>Accessing Screen Information .....</b>	<b>510</b>
<b>8.7</b>	<b>Summary .....</b>	<b>511</b>
<b>9</b>	<b>Dynamically Reloading Contents of a Web Page .....</b>	<b>513</b>
<b>9.1</b>	<b>The Principle of Ajax .....</b>	<b>513</b>
9.1.1	Synchronous Communication .....	513
9.1.2	Asynchronous Communication .....	514
9.1.3	Typical Use Cases for Ajax .....	516
9.1.4	Data Formats Used .....	518
<b>9.2</b>	<b>The XML Format .....</b>	<b>519</b>
9.2.1	The Structure of XML .....	519
9.2.2	XML and the DOM API .....	521
9.2.3	Converting Strings to XML Objects .....	522
9.2.4	Converting XML Objects to Strings .....	523
<b>9.3</b>	<b>The JSON Format .....</b>	<b>524</b>
9.3.1	The Structure of JSON .....	524
9.3.2	Difference between JSON and JavaScript Objects .....	526
9.3.3	Converting Objects to JSON Format .....	527
9.3.4	Converting Objects from JSON Format .....	528
<b>9.4</b>	<b>Making Requests via Ajax .....</b>	<b>529</b>
9.4.1	The XMLHttpRequest Object .....	529
9.4.2	Loading HTML Data via Ajax .....	535
9.4.3	Loading XML Data via Ajax .....	539

9.4.4	Loading JSON Data via Ajax .....	543
9.4.5	Sending Data to the Server via Ajax .....	545
9.4.6	Submitting DORMS via Ajax .....	546
9.4.7	Loading Data from Other Domains .....	547
9.4.8	The Newer Alternative to XMLHttpRequest: The Fetch API .....	550
<b>9.5</b>	<b>Summary .....</b>	<b>554</b>
<b>10</b>	<b>Simplifying Tasks with jQuery .....</b>	<b>555</b>
<b>10.1</b>	<b>Introduction .....</b>	<b>555</b>
10.1.1	Embedding jQuery .....	556
10.1.2	Embedding jQuery via a Content Delivery Network .....	557
10.1.3	Using jQuery .....	558
10.1.4	Simplifying Tasks with jQuery .....	559
<b>10.2</b>	<b>Working with the DOM .....</b>	<b>560</b>
10.2.1	Selecting Elements .....	561
10.2.2	Accessing and Modifying Content .....	566
10.2.3	Filtering Selected Elements .....	569
10.2.4	Accessing Attributes .....	571
10.2.5	Accessing CSS Properties .....	572
10.2.6	Navigating between Elements .....	573
10.2.7	Using Effects and Animations .....	575
<b>10.3</b>	<b>Responding to Events .....</b>	<b>576</b>
10.3.1	Registering Event Listeners .....	577
10.3.2	Responding to General Events .....	578
10.3.3	Responding to Mouse Events .....	579
10.3.4	Responding to Keyboard Events .....	581
10.3.5	Responding to Form Events .....	581
10.3.6	Accessing Information from Events .....	582
<b>10.4</b>	<b>Creating Ajax Requests .....</b>	<b>584</b>
10.4.1	Creating Ajax Requests .....	584
10.4.2	Responding to Events .....	587
10.4.3	Loading HTML Data via Ajax .....	588
10.4.4	Loading XML Data via Ajax .....	589
10.4.5	Loading JSON Data via Ajax .....	590
<b>10.5</b>	<b>Summary .....</b>	<b>592</b>

<b>11</b>	<b>Dynamically Creating Images and Graphics</b>	599
<b>11.1</b>	<b>Drawing Images</b>	599
11.1.1	The Drawing Area	599
11.1.2	The Rendering Context	600
11.1.3	Drawing Rectangles	602
11.1.4	Using Paths	604
11.1.5	Drawing Texts	610
11.1.6	Drawing Gradients	611
11.1.7	Saving and Restoring the Canvas State	613
11.1.8	Using Transformations	615
11.1.9	Creating Animations	618
<b>11.2</b>	<b>Integrating Vector Graphics</b>	620
11.2.1	The SVG Format	620
11.2.2	Integrating SVG in HTML	621
11.2.3	Changing the Appearance of SVG Elements with CSS	624
11.2.4	Manipulating the Behavior of SVG Elements via JavaScript	625
<b>11.3</b>	<b>Summary</b>	627
<b>12</b>	<b>Using Modern Web APIs</b>	629
<b>12.1</b>	<b>Communicating via JavaScript</b>	631
12.1.1	Unidirectional Communication with the Server	631
12.1.2	Bidirectional Communication with a Server	633
12.1.3	Outgoing Communication from the Server	635
<b>12.2</b>	<b>Recognizing Users</b>	639
12.2.1	Using Cookies	639
12.2.2	Creating Cookies	641
12.2.3	Reading Cookies	642
12.2.4	Example: Shopping Cart Based on Cookies	644
12.2.5	Disadvantages of Cookies	647
<b>12.3</b>	<b>Using the Browser Storage</b>	647
12.3.1	Storing Values in the Browser Storage	648
12.3.2	Reading Values from the Browser Storage	649
12.3.3	Updating Values in the Browser Storage	649
12.3.4	Deleting Values from the Browser Storage	650
12.3.5	Responding to Changes in the Browser Storage	650
12.3.6	The Different Types of Browser Storage	651
12.3.7	Example: Shopping Cart Based on the Browser Storage	653

<b>12.4</b>	<b>Using the Browser Database</b>	654
12.4.1	Opening a Database	655
12.4.2	Creating a Database	656
12.4.3	Creating an Object Store	657
12.4.4	Adding Objects to an Object Store	657
12.4.5	Reading Objects from an Object Store	661
12.4.6	Deleting Objects from an Object Store	661
12.4.7	Updating Objects in an Object Store	663
12.4.8	Using a Cursor	664
<b>12.5</b>	<b>Accessing the File System</b>	665
12.5.1	Selecting Files via File Dialog	666
12.5.2	Selecting Files via Drag and Drop	667
12.5.3	Reading Files	668
12.5.4	Monitoring the Reading Progress	671
<b>12.6</b>	<b>Moving Components of a Web Page</b>	673
12.6.1	Events of a Drag-and-Drop Operation	673
12.6.2	Defining Movable Elements	674
12.6.3	Moving Elements	676
<b>12.7</b>	<b>Parallelizing Tasks</b>	678
12.7.1	The Principle of Web Workers	679
12.7.2	Use Web Workers	680
<b>12.8</b>	<b>Determining the Location of Users</b>	682
12.8.1	Accessing Location Information	682
12.8.2	Continuously Accessing Location Information	684
12.8.3	Showing the Position on a Map	685
12.8.4	Showing Directions	686
<b>12.9</b>	<b>Reading the Battery Level of an End Device</b>	688
12.9.1	Accessing Battery Information	688
12.9.2	Responding to Events	689
<b>12.10</b>	<b>Outputting Speech and Recognizing Speech</b>	691
12.10.1	Outputting Speech	692
12.10.2	Recognizing Speech	694
<b>12.11</b>	<b>Creating Animations</b>	695
12.11.1	Using the API	696
12.11.2	Controlling an Animation	699
<b>12.12</b>	<b>Working with the Command Line</b>	699
12.12.1	Selecting and Inspecting DOM Elements	701
12.12.2	Events Analysis	704
12.12.3	Debugging, Monitoring, and Profiling	704

<b>12.13 Developing Multilingual Applications</b>	708
12.13.1 Explanation of Terms	709
12.13.2 The Internationalization API	710
12.13.3 Comparing Character String Expressions	712
12.13.4 Formatting Dates and Times	714
12.13.5 Formatting Numeric Values	717
<b>12.14 Overview of Various Web APIs</b>	720
<b>12.15 Summary</b>	724
<b>13 Object-Oriented Programming</b>	725
<b>13.1 The Principles of Object-Oriented Programming</b>	725
13.1.1 Classes, Object Instances, and Prototypes	726
13.1.2 Principle 1: Define Abstract Behavior	728
13.1.3 Principle 2: Encapsulate Condition and Behavior	728
13.1.4 Principle 3: Inherit Condition and Behavior	729
13.1.5 Principle 4: Accept Different Types	731
13.1.6 JavaScript and Object Orientation	731
<b>13.2 Prototypical Object Orientation</b>	732
13.2.1 The Concept of Prototypes	732
13.2.2 Deriving from Objects	733
13.2.3 Inheriting Methods and Properties	733
13.2.4 Defining Methods and Properties in the Inheriting Object	734
13.2.5 Overwriting Methods	735
13.2.6 The Prototype Chain	736
13.2.7 Calling Methods of the Prototype	737
13.2.8 Prototypical Object Orientation and the Principles of Object Orientation	738
<b>13.3 Pseudoclassical Object Orientation</b>	739
13.3.1 Defining Constructor Functions	739
13.3.2 Creating Object Instances	739
13.3.3 Defining Methods	740
13.3.4 Deriving from Objects	740
13.3.5 Calling the Constructor of the "Superclass"	744
13.3.6 Overwriting Methods	744
13.3.7 Calling Methods of the "Superclass"	745
13.3.8 Pseudoclassical Object Orientation and the Principles of Object Orientation	745

<b>13.4 Object Orientation with Class Syntax</b>	745
13.4.1 Defining Classes	746
13.4.2 Creating Object Instances	748
13.4.3 Defining Getters and Setters	748
13.4.4 Defining Private Properties and Private Methods	750
13.4.5 Deriving from "Classes"	753
13.4.6 Overwriting Methods	757
13.4.7 Calling Methods of the "Superclass"	759
13.4.8 Defining Static Methods	760
13.4.9 Defining Static Properties	762
13.4.10 Class Syntax and the Principles of Object Orientation	763
<b>13.5 Summary</b>	764
<b>14 Functional Programming</b>	765
<b>14.1 Principles of Functional Programming</b>	765
<b>14.2 Imperative Programming and Functional Programming</b>	767
14.2.1 Iterating with the <code>forEach()</code> Method	767
14.2.2 Mapping Values with the <code>map()</code> Method	770
14.2.3 Filtering Values with the <code>filter()</code> Method	771
14.2.4 Reducing Multiple Values to One Value with the <code>reduce()</code> Method	773
14.2.5 Combination of the Different Methods	775
<b>14.3 Summary</b>	776
<b>15 Correctly Structuring the Source Code</b>	779
<b>15.1 Avoiding Name Conflicts</b>	779
<b>15.2 Defining and Using Modules</b>	783
15.2.1 The Module Design Pattern	783
15.2.2 The Revealing Module Design Pattern	786
15.2.3 AMD	791
15.2.4 CommonJS	792
15.2.5 Native Modules	794
<b>15.3 Summary</b>	797

16

Using Asynchronous Programming and Other Advanced Features

799

16.1

Understanding and Using Asynchronous Programming

799

16.1.1

Using the Callback Design Pattern

800

16.1.2

Using Promises

804

16.1.3

Using Async Functions

813

16.2

Encapsulating Iteration over Data Structures

816

16.2.1

The Principle of Iterators

816

16.2.2

Using Iterators

817

16.2.3

Creating Your Own Iterator

817

16.2.4

Creating an Iterable Object

819

16.2.5

Iterating over Iterable Objects

820

16.3

Pausing and Resuming Functions

820

16.3.1

Creating a Generator Function

821

16.3.2

Creating a Generator

821

16.3.3

Iterating over Generators

822

16.3.4

Creating Infinite Generators

822

16.3.5

Controlling Generators with Parameters

823

16.4

Intercepting Access to Objects

824

16.4.1

The Principle of Proxies

824

16.4.2

Creating Proxies

825

16.4.3

Defining Handlers for Proxies

825

16.5

Summary

829

17

Creating Server-Based Applications with Node.js

831

17.1

Introduction to Node.js

831

17.1.1

The Architecture of Node.js

831

17.1.2

Installing Node.js

833

17.1.3

A Simple Application

833

17.2

Managing Node.js Packages

834

17.2.1

Installing the Node.js Package Manager

834

17.2.2

Installing Packages

835

17.2.3

Creating Your Own Packages

838

17.3

Processing and Triggering Events

841

17.3.1

Triggering and Intercepting an Event

841

17.3.2

Triggering an Event Multiple Times

844

20

Contents

17.3.3

Intercepting an Event Exactly Once

844

17.3.4

Intercepting an Event Multiple Times

845

17.4

Accessing the File System

846

17.4.1

Reading Files

846

17.4.2

Writing Files

847

17.4.3

Reading File Information

848

17.4.4

Deleting Files

849

17.4.5

Working with Directories

849

17.5

Creating a Web Server

851

17.5.1

Starting a Web Server

851

17.5.2

Making Files Available via Web Server

852

17.5.3

Creating a Client for a Web Server

853

17.5.4

Defining Routes

853

17.5.5

Using the Express.js Web Framework

854

17.6

Accessing Databases

859

17.6.1

MongoDB Installation

859

17.6.2

Installing a MongoDB Driver for Node.js

860

17.6.3

Establishing a Connection to the Database

860

17.6.4

Creating a Collection

861

17.6.5

Saving Objects

862

17.6.6

Reading Objects

862

17.6.7

Updating Objects

865

17.6.8

Deleting Objects

865

17.7

Working with Streams

866

17.7.1

Introduction and Types of Streams

866

17.7.2

Stream Use Cases

867

17.7.3

Reading Data with Streams

868

17.7.4

Writing Data with Streams

869

17.7.5

Combining Streams Using Piping

870

17.7.6

Error Handling during Piping

873

17.8

Summary

874

18

Creating Mobile Applications with JavaScript

877

18.1

The Different Types of Mobile Applications

877

18.1.1

Native Applications

877

18.1.2

Mobile Web Applications

878

18.1.3

Hybrid Applications

879

18.1.4

Comparison of the Different Approaches

881

21



<b>18.2</b>	<b>Creating Mobile Applications with React Native</b>	883
18.2.1	The Principle of React Native	883
18.2.2	Installation and Project Initialization	883
18.2.3	Starting the Application	884
18.2.4	The Basic Structure of a React Native Application	887
18.2.5	Using UI Components	889
18.2.6	Communication with the Server	894
18.2.7	Building and Publishing Applications	895
<b>18.3</b>	<b>Summary</b>	895
 <b>19 Desktop Applications with JavaScript</b>		897
<b>19.1</b>	<b>NW.js</b>	898
19.1.1	Installing and Creating an Application	899
19.1.2	Starting the Application	900
19.1.3	Packaging of the Application	901
19.1.4	More Sample Applications	902
<b>19.2</b>	<b>Electron</b>	903
19.2.1	Installing and Creating an Application	904
19.2.2	Starting the Application	906
19.2.3	Packaging	906
19.2.4	More Sample Applications	907
<b>19.3</b>	<b>Summary</b>	908
 <b>20 Controlling Microcontrollers with JavaScript</b>		909
<b>20.1</b>	<b>Espruino</b>	910
20.1.1	Technical Information	910
20.1.2	Connection and Installation	911
20.1.3	First Example	911
20.1.4	Controlling LEDs	912
20.1.5	More Modules	914
20.1.6	Reading Sensors	915
<b>20.2</b>	<b>Tessel</b>	916
20.2.1	Technical Information	916
20.2.2	Connection and Installation	917

20.2.3	Controlling LEDs	917
20.2.4	Programming the Push Buttons	919
20.2.5	Extending the Tessel with Modules	920
<b>20.3</b>	<b>BeagleBone Black</b>	921
20.3.1	Technical Information	921
20.3.2	Connection and Installation	922
20.3.3	Controlling LEDs	923
<b>20.4</b>	<b>Arduino</b>	924
20.4.1	The Firmata Protocol	924
20.4.2	Connection and Installation	925
20.4.3	The Johnny Five Node.js Module	925
<b>20.5</b>	<b>Cylon.js</b>	927
20.5.1	Controlling the BeagleBone Black with Cylon.js	927
20.5.2	Controlling the Tessel Board with Cylon.js	928
20.5.3	Controlling an Arduino with Cylon.js	928
<b>20.6</b>	<b>Summary</b>	929
 <b>21 Establishing a Professional Development Process</b>		931
<b>21.1</b>	<b>Automating Tasks</b>	931
21.1.1	Automating Tasks with Grunt	931
21.1.2	Automating Tasks with Gulp	935
<b>21.2</b>	<b>Automated Testing of Source Code</b>	936
21.2.1	The Principle of Automated Tests	936
21.2.2	The Principle of Test-Driven Development	937
21.2.3	Automated Testing of Source Code with QUnit	939
21.2.4	Automated Testing of Source Code with mocha	945
<b>21.3</b>	<b>Source Code Version Management</b>	949
21.3.1	Introduction to Version Management	949
21.3.2	Installing and Configuring the Git Version Control System	953
21.3.3	Creating a New Local Repository	954
21.3.4	Cloning an Existing Repository	954
21.3.5	Transferring Changes to the Staging Area	955
21.3.6	Transferring Changes to the Local Repository	956
21.3.7	The Different States in Git	957
21.3.8	Transferring Changes to the Remote Repository	958
21.3.9	Transferring Changes from the Remote Repository	959

21.3.10 Working in a New Branch .....	960
21.3.11 Adopting Changes from a Branch .....	961
21.3.12 Overview of the Most Important Commands and Terms .....	962
<b>21.4 Summary</b> .....	965
The Author .....	967
Index .....	969

# Index

__proto__	241, 732
:invalid	477
:required	477
:valid	477
<canvas>	599
<noscript>	70
<script>	70
\$()	558

## A

Abstraction	725, 728
Access property	247, 255, 739
Accessing attributes	571
Accessing CSS properties	572
Accessor	729
Accessor method	729
Actuator	909
addEventListener()	417
Addition	114
Aggregation	726
Ajax	48, 513, 584
alert()	66, 76
Algorithm	32
AMD	791
Analyze events	704
AND operator	117
Android	51, 877
Animation	696
<i>AnimationEffectTiming</i>	696
<i>AnimationTimeline</i>	696
<i>cancel</i>	699
<i>pause</i>	699
<i>start</i>	699
Anonymous function	169
Answer	513
App	877
appendChild()	394
Apple	877
Application	32
<i>hybrid</i>	52
<i>native</i>	36, 51
application/ecmascript	68
application/javascript	68
apply()	356
Arduino	924
Argument	172, 173

Array	104, 270
<i>access elements</i>	273
<i>add elements</i>	274
<i>convert to strings</i>	293
<i>copy elements</i>	282, 292
<i>create</i>	270
<i>elements</i>	104
<i>entries</i>	104
<i>extract elements</i>	294
<i>find elements</i>	290
<i>index-based structure</i>	106
<i>initialize</i>	270
<i>multidimensional</i>	106
<i>remove elements</i>	279
<i>sort</i>	284
<i>use as queue</i>	288
<i>use as stack</i>	287
<i>values</i>	104
Array destructuring	294
Array literal notation	104, 270, 271
Array.isArray()	235
Arrow function	186
Assembly language	33
Assertion	936
Assignment operator	86, 115
Association	726
Async functions	72, 813
Asynchronous JavaScript and XML	48, 513
Asynchronous module definition	783
Asynchronous programming	799
Atom	60
attachEvent()	418
Attribute	109
Attribute node	360
Automatic builds	61

## B

Backend	48
Bad practices	48
Battery Status API	688
BeagleBone Black	921
<i>control LEDs</i>	923
Behavior	726
Behavioral layer	58
Best practices	48
Bidirectional communication	631

Binary buffer ..... 868  
Binary code ..... 33  
Binary Large Object ..... 634  
Binary notation ..... 96  
Binary number ..... 96  
bind() ..... 353  
Blob ..... 634, 665  
Blocking I/O ..... 831  
Boilerplate code ..... 767  
BOM API ..... 487  
BoneScript ..... 922  
Boolean ..... 103  
Boolean value ..... 103  
Bootstrap ..... 462  
Branch ..... 133, 134, 960  
break ..... 160  
Breakpoint ..... 218  
    *conditional* ..... 223  
    *DOM* ..... 224  
    *event listener* ..... 224  
    *XHR* ..... 224  
Browser  
    *address bar* ..... 493  
    *menu bar* ..... 493  
    *personal bar* ..... 493  
    *recognize* ..... 508  
    *scrollbars* ..... 493  
    *status bar* ..... 493  
    *toolbar* ..... 493  
Browser database ..... 654  
Browser detection ..... 509  
Browser feature ..... 508  
Browser Object Model ..... 487  
Browser sniffing ..... 509  
Browser storage ..... 647  
Browser window  
    *change position* ..... 490  
    *close* ..... 495  
    *determine general properties* ..... 493  
    *determine position* ..... 489  
    *determine size* ..... 489  
    *open new* ..... 494  
    *resize* ..... 490  
Browsing history ..... 502  
    *replace current entry* ..... 506  
    *respond to changes* ..... 506  
    *state object* ..... 504  
Bubbling phase ..... 439  
Bug ..... 200  
Build tool ..... 931  
Bytecode ..... 39

## C

Cache ..... 85  
call() ..... 355  
Callback ..... 800  
Callback design pattern ..... 800  
Callback function ..... 769, 800  
Callback handler ..... 800  
Callback hell ..... 803  
CamelCase spelling ..... 92  
Canvas API ..... 430, 599  
    *2D rendering context* ..... 600  
    *3D rendering context* ..... 600  
    *drawing arcs and circles* ..... 608  
    *drawing area* ..... 599  
    *drawing Bezier curves* ..... 607  
    *drawing rectangles* ..... 602  
    *drawing square curves* ..... 607  
    *drawing texts* ..... 610  
    *rendering context* ..... 600  
    *rotation* ..... 616  
    *scaling* ..... 615  
    *translation* ..... 617  
    *using paths* ..... 604  
CanvasRenderingContext2D ..... 601  
Capturing phase ..... 439  
Cascading Style Sheets ..... 51  
catch ..... 202  
Character class ..... 333  
    *negation* ..... 333  
    *predefined* ..... 334  
    *range* ..... 333  
    *simple class* ..... 333  
Checkbox, read ..... 467  
checkValidity() ..... 482  
Chrome DevTools ..... 78, 217, 361  
Class ..... 242, 726  
    *defining* ..... 746  
    *deriving* ..... 730  
    *that inherits* ..... 730  
Class body ..... 243, 746  
Class declaration ..... 746  
Class diagram ..... 236, 726  
Class expressions ..... 747  
Class that inherits ..... 730  
Class under test ..... 936  
Class-based object orientation ..... 727  
Class-based programming language ..... 727  
Classless programming ..... 727  
classList ..... 407  
className ..... 407  
Client ..... 513

Client side ..... 48  
Closure ..... 783  
Cloud9 IDE ..... 922  
CLR ..... 38  
Code, global ..... 191  
Collator ..... 712  
Collection ..... 861  
Column families ..... 654  
Command line ..... 699  
Command Line API ..... 700  
    *Debugging* ..... 704  
    *Monitoring* ..... 704  
    *Profiling* ..... 704  
Comment ..... 216  
    *multi-line* ..... 216  
Common Language Runtime ..... 38  
CommonJS ..... 783, 792  
Communication  
    *asynchronous* ..... 514  
    *bidirectional* ..... 631  
    *synchronous* ..... 513  
    *unidirectional* ..... 631  
Comparing character string expressions ..... 712  
Compiler ..... 35  
Component test ..... 938  
Composition ..... 726  
Concatenation ..... 116  
configurable ..... 247  
confirm() ..... 76  
Confirmation dialog ..... 76  
Console ..... 78, 79  
    *show in Chrome* ..... 78  
    *show in Firefox* ..... 78  
    *show in Opera* ..... 78  
    *show in Safari* ..... 78  
const ..... 94  
Constant ..... 94  
    *define* ..... 94  
Constraint Validation API ..... 478  
Constructor ..... 748  
    *constructor* ..... 241  
    *constructor function* ..... 239, 270, 739  
    *constructor()* ..... 243, 747  
Content delivery network ..... 557  
Content distribution network ..... 557  
Content layer ..... 58  
Content type ..... 68  
Context object ..... 192  
continue ..... 160, 162  
Control character ..... 100  
Control structure ..... 133

Convert string  
    *to XML objects* ..... 522  
Cookie ..... 639, 641  
CORS ..... 549  
CouchDB ..... 654  
Counter variable ..... 148  
Counting loop ..... 148  
Coupling ..... 726  
Create object  
    *classes* ..... 242  
    *constructor functions* ..... 239  
    *literal notation* ..... 237  
    *Object.create()* ..... 246  
createAttribute() ..... 403, 406  
createElement() ..... 394  
createTextNode() ..... 393  
Creating Ajax requests ..... 584  
Cross-origin request ..... 547  
    *proxy* ..... 548  
Cross-Origin Resource Sharing ..... 549  
CRUD ..... 654  
CSS ..... 55  
    *class* ..... 367  
CSS3 ..... 51  
    *media queries* ..... 878  
Cursor ..... 664  
CustomEvent ..... 455  
CVS ..... 949  
Cylon.js ..... 927

## D

Data encapsulation ..... 252, 725, 728, 782  
Data property ..... 255, 739  
Data structure ..... 358  
Data type ..... 94, 95  
    *primitive* ..... 94  
    *special* ..... 110  
Data URL ..... 669  
Data, numeric ..... 95  
Database ..... 654  
    *create* ..... 656  
    *document-oriented* ..... 859  
    *transaction* ..... 658  
Database schema ..... 655, 657  
Date ..... 325  
DateFormat ..... 714  
Debugging ..... 80, 216, 217, 219  
Decision point ..... 43  
Declarative ..... 766  
Decrement operator ..... 114  
Default parameter ..... 182

Default value .....	182	DOM (Cont.)	
Defensive programming .....	365	<i>read attributes</i> .....	403
defer .....	72	<i>remove attributes</i> .....	406
Define animations .....	695	<i>Scripting</i> .....	48
Define movable element .....	674	<i>tree</i> .....	357
Delete .....	260	DOMContentLoaded .....	417
Deriving class .....	730	DOMParser .....	522
Design pattern .....	783	Dot notation .....	249
Desktop applications .....	897	Drag and Drop API .....	667, 673
Destructuring .....	294	Drag source .....	673
Destructuring statement .....	294	Drag-and-drop operation .....	673
Determine		<i>events</i> .....	673
<i>date</i> .....	326	Drawing a Bézier curve .....	607
<i>day of month</i> .....	326	Drawing arcs and circles .....	608
<i>hour</i> .....	326	Drawing area .....	599
<i>minute</i> .....	326	Drawing text .....	610
<i>month</i> .....	326	Drop target .....	673
<i>seconds</i> .....	326	DTD .....	519
<i>the current month</i> .....	325	Duplex streams .....	867
<i>time</i> .....	326	Dynamic HTML .....	48
<i>weekday</i> .....	326		
<i>year</i> .....	326	<b>E</b>	
Development environment .....	61	Easing function .....	575
Development, test-driven .....	937	ECMAScript .....	46
Directory		Editor .....	59
<i>create</i> .....	849	Eich, Brendan .....	46
<i>delete</i> .....	849	Electron .....	903
<i>list files</i> .....	849	<i>installation</i> .....	904
dispatchEvent() .....	454	<i>packaging</i> .....	906
Diversity .....	731	Element	
Division .....	114	<i>filter</i> .....	569
Document .....	359	<i>getAttribute()</i> .....	403
Document database .....	654	<i>move</i> .....	676
Document node .....	359, 361	<i>navigate between</i> .....	573
Document Object Model .....	357	<i>node</i> .....	359
Document type definition .....	519	<i>removeAttribute()</i> .....	403
document.anchors .....	390	<i>select</i> .....	363, 561
document.body .....	390	<i>select by class</i> .....	367
document.forms .....	390	<i>select by element name</i> .....	370
document.head .....	390	<i>select by ID</i> .....	364
document.images .....	390	<i>select by name</i> .....	371
document.links .....	390	<i>select by selector</i> .....	373
DocumentTimeline .....	696	<i>setAttribute()</i> .....	403
DOM .....	357	Encapsulation .....	252, 728
<i>access CSS classes</i> .....	406	enumerable .....	247
<i>API</i> .....	362	Error .....	203
<i>change the value of an attribute</i> .....	405	<i>catch and handle</i> .....	202
<i>create and add attribute nodes</i> .....	406	<i>logic</i> .....	200
<i>event handler</i> .....	411	<i>trigger</i> .....	205
<i>event listener</i> .....	411	Error handling .....	201
<i>examine in browser</i> .....	361	Escape character .....	99
<i>manipulation</i> .....	392		

Escaping .....	99	Execution context .....	191
Espruino .....	52, 910	<i>global</i> .....	191
<i>control LEDs</i> .....	912	Exponential operator .....	114
<i>HTTP client</i> .....	914	export .....	794
<i>HTTP server</i> .....	914	Export object .....	785
<i>read files</i> .....	914	Extensible Markup Language .....	518, 519
<i>read sensors</i> .....	915	Extension sub tags .....	710
<i>Web IDE</i> .....	911		
<i>write files</i> .....	914	<b>F</b>	
European Computer Manufacturers		False .....	103
Association .....	46	Falsy .....	119
eval() .....	191	Fat arrow function .....	186
EvalError .....	203	Feature detection .....	509, 602
Event .....	409	Fetch API .....	550, 894
<i>access information of the event</i> .....	424	File .....	665
<i>bind</i> .....	410	<i>delete</i> .....	849
<i>bubbling</i> .....	440, 441	<i>read</i> .....	846
<i>capturing</i> .....	440, 445	<i>write</i> .....	847
<i>emitter</i> .....	409, 841	FileList .....	665
<i>focus events</i> .....	435	FileReader .....	665, 668
<i>form events</i> .....	434	filter() .....	771
<i>intercept</i> .....	841	finally .....	210
<i>keyboard events</i> .....	431	find() .....	292, 775
<i>loop</i> .....	409, 832	findIndex() .....	292
<i>method</i> .....	577	Firebug .....	79
<i>mobile devices</i> .....	438	Firmata protocol .....	924
<i>mouse events</i> .....	427	First in, first out .....	288
<i>phase</i> .....	439	firstChild .....	383
<i>prevent default actions</i> .....	452	first-class citizen .....	765
<i>queue</i> .....	409	First-class object .....	765
<i>respond to events</i> .....	410	firstElementChild .....	383
<i>trigger</i> .....	409, 454, 841	Flag .....	344
<i>trigger programmatically</i> .....	454	Flowchart .....	41
<i>user interface events</i> .....	435	Flowchart notation .....	41
Event flow .....	439	Fluent API .....	581
<i>interrupt</i> .....	447	Focus event .....	435
Event handler .....	73, 409	FocusEvent .....	435
<i>define via HTML</i> .....	412	for loop .....	149
<i>define via JavaScript</i> .....	415	forEach() .....	351, 767
<i>helper function</i> .....	423	for-in loop .....	247, 262, 317
Event listener .....	73	Form	
<i>define</i> .....	417	<i>access to</i> .....	459
<i>define multiple</i> .....	418	<i>element access</i> .....	463
<i>helper function</i> .....	423	<i>event</i> .....	434, 581
<i>pass arguments</i> .....	420	<i>reset</i> .....	472
<i>register</i> .....	577	<i>submit</i> .....	472
<i>remove</i> .....	422	<i>validate</i> .....	475
every() .....	775	Formatting dates and times .....	714
Exception .....	201	Formatting numeric values .....	717
Exception Handling .....	201	FormData .....	546
Executable machine code file .....	35	for-of loop .....	317

Fragment identifier .....	499	Git (Cont.)	
Front end .....	48	<i>index</i> .....	952, 965
Function .....	66, 171	<i>local working copy</i> .....	952
<i>anonymous</i> .....	169	<i>merge</i> .....	965
<i>borrowing</i> .....	355	<i>pull</i> .....	965
<i>call</i> .....	171	<i>push</i> .....	965
<i>call stack</i> .....	190	<i>remote repository</i> .....	965
<i>call with a parameter</i> .....	172	<i>repository</i> .....	965
<i>call with multiple parameters</i> .....	174	<i>snapshot</i> .....	951
<i>closure</i> .....	783	<i>staging area</i> .....	952, 965
<i>code</i> .....	191	<i>working directory</i> .....	952, 965
<i>declaration</i> .....	169	<i>workspace</i> .....	965
<i>define</i> .....	168	Global jQuery method .....	585
<i>define with a parameter</i> .....	172	Google Maps API .....	685
<i>define with multiple parameters</i> .....	173	Gradient	
<i>expression</i> .....	169	<i>linear</i> .....	611
<i>methods</i> .....	353	<i>radial</i> .....	611
<i>named</i> .....	169	Grammar .....	85
<i>parameters</i> .....	172	Graph database .....	654
<i>signature</i> .....	173	Grunt .....	931
<i>variadic</i> .....	178	<i>installing plug-ins</i> .....	933
Functional Programming .....	765	<i>using plug-ins</i> .....	933
Functional programming		Gruntfile.js .....	932
<i>difference to imperative programming</i> ...	767	Gulp .....	935
<i>filter values</i> .....	771	gulpfile.js .....	935
<i>iterate</i> .....	767		
<i>language</i> .....	765		
<i>map values</i> .....	770		
<i>reduce multiple values to one value</i> .....	773		
<b>G</b>		<b>H</b>	
Garbage collection .....	319, 324	Hashbang URL .....	503
General event .....	578	Hello World example .....	64
Generator .....	820	Hexadecimal notation .....	96
<i>infinite</i> .....	822	Hexadecimal number .....	96
<i>parameter</i> .....	823	High-level programming language .....	33
Geolocation .....	682	Hint dialog .....	76
<i>API</i> .....	682	History .....	46, 503
get .....	251	<i>pushState()</i> .....	504
getElementById() .....	364	<i>replaceState()</i> .....	504, 506
getElementsByClassName() .....	367	history .....	502
getElementsByName() .....	372	History API .....	503
getElementsByTagName() .....	370	HTML .....	55
Getter method .....	251, 785	HTML element .....	66, 398
Git .....	950	HTML event handler .....	411
<i>branch</i> .....	965	HTML5 app .....	878
<i>checkout</i> .....	965	HTML5 application .....	51
<i>clone</i> .....	965	HTTP .....	513
<i>commit</i> .....	965	HTTP protocol .....	639
<i>fork</i> .....	965	HTTP request .....	513
<i>HEAD</i> .....	965	HTTP response .....	513
		Hybrid app .....	879
		Hybrid application .....	879
		Hypertext Transfer Protocol .....	513

<b>I</b>		iPad .....	877
i18n .....	708	iPhone .....	877
IDE .....	61	isSealed() .....	268
if statement .....	133	Iterable .....	820
if-else-if-else branch .....	136	Iteration .....	148
Immediately Invoked Function		Iterator .....	317, 816
Expression (IIFE) .....	783		
Imperative programming .....	766	<b>J</b>	
Implementation .....	41, 362	Java .....	46, 877
import() .....	796	Java Runtime Environment (JRE) .....	38
Increment operator .....	114	JavaScript library .....	77
Indexed Database API .....	654	JavaScript Object Notation .....	519, 524
IndexedDB API .....	654	JIT .....	38
indexOf() .....	290, 308	Johnny Five .....	52, 925
Infinite loop .....	164	jQuery	
Infinity .....	98	<i>access attributes</i> .....	571
Information hiding .....	728, 782	<i>access content</i> .....	566
Inheritance .....	265, 726, 729	<i>access CSS properties</i> .....	572
innerHTML .....	392	<i>Ajax</i> .....	584
innerText .....	392	<i>Ajax events</i> .....	587
Input .....	43	<i>attribute filter selectors</i> .....	561, 563
Input and output		<i>basic filter selectors</i> .....	561, 562
<i>blocking</i> .....	831	<i>basic selectors</i> .....	561
<i>non-blocking</i> .....	833	<i>child filter selectors</i> .....	561, 564
Input dialog .....	76	<i>content filter selectors</i> .....	561, 563
insertBefore() .....	394	<i>embed</i> .....	556
Instance .....	726	<i>embed via a CDN</i> .....	557
Interface .....	362	<i>event methods</i> .....	577
Interface element .....	50	<i>event object</i> .....	582
Intermediate language .....	35	<i>filter elements</i> .....	569
Internationalisierung		<i>form events</i> .....	581
<i>locales</i> .....	709	<i>form filter selectors</i> .....	561, 564
<i>sub tags</i> .....	709	<i>general events</i> .....	578
Internationalization		<i>global jQuery methods</i> .....	585
<i>extension subtag</i> .....	710	<i>hierarchy selectors</i> .....	561, 562
<i>language subtag</i> .....	709	<i>keyboard events</i> .....	581
<i>language tag</i> .....	709	<i>load HTML data via Ajax</i> .....	588
<i>region subtag</i> .....	709	<i>load JSON data via Ajax</i> .....	590
<i>script subtag</i> .....	710	<i>load XML data via Ajax</i> .....	589
<i>variant subtag</i> .....	710	<i>modify content</i> .....	566
Internationalization (i18n) .....	708	<i>mouse events</i> .....	579
Internationalization API .....	710	<i>navigate between elements</i> .....	573
Internet media type .....	68	<i>register event listener</i> .....	577
Internet of Things (IoT) .....	52, 909	<i>respond to events</i> .....	576
Interpret .....	37	<i>select elements</i> .....	561
Interpreter .....	35	<i>selected elements</i> .....	559
Intl.Collator .....	711	<i>selected nodes</i> .....	559
Intl.DateTimeFormat .....	711	<i>use effects</i> .....	575
Intl.NumberFormat .....	711	<i>visibility filter selectors</i> .....	561, 563
Ionic .....	883	jQuery method .....	558
iOS .....	51	jQuery object .....	558



jQuery() ..... 558  
JRE ..... 38  
JSON ..... 519, 524, 527  
JSON format ..... 524  
JSON with padding ..... 550  
JSON.stringify() ..... 527, 528  
JSONP ..... 550  
JSX ..... 888  
Jump label ..... 165  
Just-in-Time Compiler ..... 38

K

Keyboard event ..... 431, 581  
KeyboardEvent ..... 431  
KeyframeEffect ..... 696  
Key-value pair ..... 314  
Key-value stores ..... 654  
Keyword ..... 88  
Kotlin ..... 877

L

Label ..... 165  
Language subtag ..... 709  
Language tag ..... 709  
lastChild ..... 383  
lastElementChild ..... 383  
lastIndexOf() ..... 308  
let ..... 85  
Library ..... 50  
LIFO principle ..... 287  
Literal ..... 113  
Literal notation ..... 237  
LiveScript ..... 46  
Local browser storage ..... 647  
Locales ..... 709  
localStorage ..... 647  
Location ..... 499  
    *assign()* ..... 500  
    *href* ..... 501  
    *reload()* ..... 501  
    *replace()* ..... 501  
Logical AND assignment operator ..... 130  
Logical assignment operators ..... 130  
Logical nullish assignment operator ..... 130  
Logical OR assignment operator ..... 130  
Long polling ..... 632  
Look-and-feel ..... 883  
Loop ..... 133, 148  
    *head-controlled* ..... 148, 155  
    *inner* ..... 153

Loop (Cont.)  
    *nested* ..... 153  
    *outer* ..... 153  
    *tail-controlled* ..... 149, 158  
    *terminate prematurely* ..... 160  
Loop body ..... 149, 155  
Loose augmentation ..... 790  
Loose typing ..... 95  
lowerCamelCase notation ..... 92

M

Machine code ..... 33  
Machine language ..... 33  
Main thread ..... 678  
Map ..... 272, 314  
    *add elements* ..... 315  
    *delete all key-value pairs* ..... 315  
    *delete individual key-value pairs* ..... 315  
    *determine elements for key* ..... 315  
map() ..... 770  
Markup language ..... 59  
Math ..... 328  
Maximum value of numbers ..... 98  
MEAN ..... 866  
    *stack* ..... 866  
Member ..... 250  
Member operator ..... 250  
Mercurial ..... 950  
Method ..... 109  
    *borrowing* ..... 355, 369  
    *call stack* ..... 80, 190  
    *signature* ..... 173  
    *static* ..... 760  
Microcontroller ..... 909  
Middleware ..... 854  
    *function* ..... 854  
Minimum value of numbers ..... 97  
Mobile application  
    *hybrid* ..... 879  
    *native* ..... 877  
Mobile web application ..... 878  
Mocha ..... 46, 945  
Modifier ..... 344  
Modifying the CSS of an element ..... 369  
Module  
    *default export* ..... 794  
    *define* ..... 794  
    *export* ..... 794  
    *import* ..... 795  
    *import dynamically* ..... 796  
    *named export* ..... 794

Module augmentation ..... 789  
Module design pattern ..... 783  
Module test ..... 938  
Modulo ..... 114  
MongoDB ..... 654, 859  
Mouse event ..... 427, 579  
MouseEvent ..... 429  
Multidimensional array ..... 106  
Multilingual applications ..... 708  
Multiple selection list ..... 470  
    *read* ..... 470  
Multiplication ..... 114  
Multipurpose Internet Mail  
    Extension (MIME) ..... 68  
Multithreaded ..... 678  
    *servers* ..... 831  
Multiway branch ..... 142

N

Named function ..... 169  
Namespace ..... 779  
Namespace design pattern ..... 779  
NaN ..... 98  
Native application ..... 877, 883  
NativeScript ..... 883  
Navigator ..... 508  
Negation operator ..... 117  
Nested namespacing ..... 782  
Netscape ..... 46  
nextElementSibling ..... 386  
nextSibling ..... 385  
Node ..... 357  
Node list ..... 368  
    *static* ..... 369  
Node type ..... 358  
Node.js ..... 357  
    *create packages* ..... 838  
    *instal packages* ..... 835  
    *install packages globally* ..... 835  
    *install packages locally* ..... 835  
    *modules* ..... 831  
    *Node.js Package Manager (NPM)* ..... 834  
    *NPM* ..... 834  
    *Package configuration file* ..... 838  
    *packages* ..... 831  
    *use packages* ..... 836  
Nonblocking I/O ..... 833  
Nonrelational database ..... 654  
NPM ..... 834  
npm init ..... 838  
npm install ..... 835

NPM Registry ..... 836  
null ..... 110  
Nullish coalescing operator ..... 122  
Number ..... 95  
    *definition* ..... 95  
    *value range* ..... 97  
NumberFormat ..... 717  
NW.js ..... 898  
    *Installation* ..... 899  
    *Packaging* ..... 901

O

Object ..... 109, 726  
    *array-like* ..... 177  
    *behavior* ..... 236  
    *bind* ..... 353  
    *definition* ..... 109  
    *extract values* ..... 298  
    *first class* ..... 765  
    *freeze* ..... 269  
    *global* ..... 488  
    *intercept access* ..... 824  
    *iterable* ..... 819  
    *Object.create()* ..... 733  
    *Object.defineProperties()* ..... 258  
    *Object.defineProperty()* ..... 258  
    *Object.freeze()* ..... 269  
    *Object.getOwnPropertyDescriptor()* ..... 248  
    *Object.getPrototypeOf()* ..... 241  
    *Object.isExtensible()* ..... 268  
    *Object.isFrozen()* ..... 269  
    *Object.preventExtensions()* ..... 267  
    *Object.seal()* ..... 268  
    *prevent extensions* ..... 267  
    *seal* ..... 268  
    *state* ..... 236  
Object destructuring ..... 298  
Object diagram ..... 726  
Object instance ..... 726  
    *create* ..... 748  
Object literal notation ..... 237  
Object method ..... 109  
    *add* ..... 256  
    *create via bracket notation* ..... 257  
    *create via dot notation* ..... 256  
    *create via helper methods* ..... 258  
    *delete* ..... 260  
    *output* ..... 262  
    *overwrite* ..... 256  
Object orientation ..... 725  
    *basic principles* ..... 725

Object orientation (Cont.)	
<i>class syntax</i>	731
<i>class-based</i>	727
<i>prototypical</i>	727, 731
<i>pseudoclassical</i>	731
Object property	109
<i>add</i>	256
<i>create via bracket notation</i>	257
<i>create via dot notation</i>	256
<i>create via helper methods</i>	258
<i>delete</i>	260
<i>output</i>	262
<i>overwrite</i>	256
Object store	
<i>add objects</i>	657
<i>create</i>	657
<i>delete objects</i>	661
<i>read objects</i>	661
<i>update objects</i>	663
<i>use cursor</i>	664
Object.entries()	238, 262, 263
Object.keys()	262, 263
Object.values()	262, 263
Object-based programming language	727
Objective-C	877
Object-oriented programming	242, 725, 766
Octal notation	96
Octal number	96
onload	416
onreadystatechange	534
Operation	43
Operator	113
<i>arithmetic</i>	114
<i>binary</i>	114, 117
<i>bitwise</i>	124
<i>logical</i>	117
<i>logical AND assignment</i>	130
<i>logical nullish assignment</i>	130
<i>logical OR assignment</i>	130
<i>overloaded</i>	117
<i>unary</i>	113, 117
Optional chaining operator	128, 366
OR operator	117
Output	43
<b>P</b>	
Package	779
Package configuration file	838
package.json	838
<i>properties</i>	839
Parameter	172, 173
Parent class	732
Parent constructor	754
parentElement	379
parentNode	379
Parsing	70
Password field, read	465
Path	604
Pattern	783
Polling	631
Polymorphic	731
Polymorphism	726, 731
pop()	279
PopStateEvent	506
Preflight request	549
Preflight response	549
Prematurely terminating loop iterations	162
Presentation Layer	58
preventDefault()	447, 452
previousElementSibling	386
previousSibling	385
Private methods	750
Private properties	750
Private property	252
Program	32
<i>native</i>	36
Program flowchart	41
Programming	
<i>asynchronous</i>	799
<i>classless</i>	727
<i>event-driven</i>	409
<i>functional</i>	765
<i>imperative</i>	766
<i>object-oriented</i>	725, 766
<i>prototype-based</i>	727
<i>prototypical</i>	727
Programming language	33, 59
<i>class-based</i>	727
<i>compiled</i>	35
<i>functional</i>	765
<i>interpreted</i>	35
<i>object-based</i>	727
ProgressEvent	671
Promise	804
Promise.all()	808
Promise.allSettled()	810
Promise.any()	811
Promise.prototype.catch()	805
Promise.prototype.finally()	806
Promise.prototype.then()	805
Promise.race()	809
Promise.reject()	812
Promise.resolve()	812

prompt()	76
Property	109
<i>static</i>	762
Property attribute	246, 739
Prototype	241, 727, 732
Prototype chain	242, 736
Prototype programming	727
Prototype-based programming	727
Prototypical object orientation	727, 731, 732
<i>calling methods of the prototype</i>	737
<i>inherit methods and properties</i>	733
<i>overwrite methods</i>	735
Proxy	824
Pseudoclassical object orientation	731, 739
<i>create object instances</i>	739
<i>define methods and properties</i>	740
<i>derive from objects</i>	740
Pseudocode	44
Pseudocode technique	41
push()	275
Pyramid of doom	803
<b>Q</b>	
Query string	499
querySelector()	373
querySelectorAll()	373
Queue	272, 288
QUnit	939
<b>R</b>	
Radio button, read	467
RangeError	203
Raspberry Pi	921
React Native	877, 883
Read file information	848
Readable stream	866, 868
Read-eval-print loop	833
Redis	654
reduce()	773
reduceRight()	775
Refactoring	938
ReferenceError	203
RegExp	
<i>exec()</i>	329, 343
<i>test()</i>	329, 330
Region subtag	709
Regular expression	
<i>define any number of occurrences</i>	339
<i>define at least one occurrence</i>	339
Regular expression (Cont.)	
<i>define exact number of occurrences</i>	340
<i>define minimum and maximum</i>	
<i>number of occurrences</i>	340
<i>define minimum number of</i>	
<i>occurrences</i>	340
<i>define optional occurrences</i>	339
<i>groups</i>	345
<i>named groups</i>	346
<i>replace occurrence</i>	347
<i>search for specific strings</i>	346
Relational database	654
Relational operator	125
Remainder operator	114
Remote branch	959
Remote repository	950
removeAttribute()	406
removeChild()	397
removeEventListener()	422
Render	59
Rendering context	600
Repetition	133
REPL	833
replaceChild()	394
reportValidity()	482
Repository	949
<i>local</i>	950
<i>remote</i>	950
Request	513
Request queue	832
requestAnimationFrame()	618
reset()	472
Responsive app	878
Responsive web design	878
Rest parameter	177
Rest properties	304
Return code	180
Return value	
<i>multiple</i>	182
Revealing module design pattern	786
Revealing module pattern	787
reverse()	284
RFC3986	670
Rich internet application	47
Root node	359
Routing	853
RTE	37
Runtime environment (RTE)	37
Runtime error	199

S

Scalable Vector Graphics ..... 599

Schema ..... 657

Scope ..... 192

*chain* ..... 192, 193

*dynamic* ..... 192

*lexical* ..... 192

*static* ..... 192

Scope chain ..... 193

Screen ..... 510

Screen Orientation API ..... 511

Script subtag ..... 710

Search string ..... 499

Secure Sockets Layer ..... 641

Selection list

*fill with values* ..... 471

*read* ..... 469

Selection operator ..... 140

Sensor ..... 909

Serialize XML data ..... 523

Server ..... 513

Server side ..... 48

Server-sent events ..... 631, 635

Session storage ..... 647

Set ..... 251, 272, 321

*add elements* ..... 321

*create* ..... 321

*delete all elements* ..... 322

*delete individual elements* ..... 322

setAttributeNode() ..... 403, 406

setInterval() ..... 498

Setter ..... 251

setTimeout() ..... 498

shift() ..... 280

Shopping cart ..... 644, 653

Side effect ..... 766

Single-page application ..... 503

Single-threaded server ..... 832

slice() ..... 282

Smartphone ..... 877

Smartwatch ..... 52

Socket connection ..... 633

Software ..... 32

Software library ..... 50

some() ..... 775

Sort array

*arrays with objects* ..... 285

*comparison function* ..... 284

sort() ..... 284

Source code ..... 34

*automated testing* ..... 936

Source code (Cont.)

*display* ..... 74

*show in Chrome* ..... 74

*show in Firefox* ..... 74

*show in Microsoft Edge* ..... 76

*show in Opera* ..... 74

*show in Safari* ..... 74

Source control system ..... 61

Source file ..... 34

Source text ..... 34

Speech output ..... 692

Speech recognition ..... 694

SpeechRecognition ..... 691

SpeechSynthesis ..... 691

splice() ..... 277, 281

Spread operator ..... 184

Spread properties ..... 305

Square curve

*drawing* ..... 607

Stack ..... 272, 287

Stack trace ..... 80

Standard variable, global ..... 90

Start/stop checkpoints ..... 42

State ..... 726

Stateless protocol ..... 639

Statement ..... 43, 73

*conditional* ..... 133

*destructuring* ..... 294

*named* ..... 166

static ..... 760

Static methods ..... 760

Static property ..... 762

Step ..... 32

stopPropagation() ..... 447

Storage ..... 647

StorageEvent ..... 650

Store value in variables ..... 85

Streams

*flowing mode* ..... 869

*paused mode* ..... 869

*pipng* ..... 870

*read data* ..... 868

*write data* ..... 869

Strict Typing ..... 95

String ..... 98, 116

*compare* ..... 712

*definition* ..... 98

*determine length* ..... 307

*escaping* ..... 99

*expressions* ..... 102

*extract parts* ..... 310

*match()* ..... 329

String (Cont.)

*multiline* ..... 102

*placeholder* ..... 101

*replace()* ..... 329, 347

*search* ..... 308

*search for occurrences* ..... 347

*search()* ..... 329

*split* ..... 348

*split()* ..... 329

*structure* ..... 306

*substr()* ..... 312

*substring()* ..... 312

String concatenation ..... 101

String operator ..... 116

Strong typing ..... 95

Style language ..... 59

Sub tags ..... 709

Subclass ..... 730

Sublime Text ..... 60

submit() ..... 472

Subobject ..... 730

Subtraction ..... 114

Subtype ..... 730

Subversion ..... 949

Sun ..... 46

Superclass ..... 730, 732

Superobject ..... 730

Supertype ..... 730

SVG format ..... 620

Swift ..... 877

Switch branch ..... 142

Symbol ..... 112

Symbol.iterator ..... 266

Syntax ..... 85

Syntax error ..... 198, 203

System under test ..... 936

T

Tablet ..... 877

Tag function ..... 188

Tagged template ..... 103, 188

Target phase ..... 439

Task ..... 932

Template string

*expressions* ..... 102

*multiline* ..... 102

*placeholder* ..... 101

*use* ..... 101

Tessel ..... 52, 916

*control LEDs* ..... 917

Test anything protocol ..... 948

Test case ..... 936

Test fixture ..... 939

Test framework ..... 61

Test program ..... 936

Test-driven development ..... 937

Text field, read ..... 465

Text node ..... 360

text/ecmascript ..... 68

text/javascript ..... 68

textContent ..... 391

this ..... 192, 194

*constructor function* ..... 196

*global function* ..... 196

*object method* ..... 196

Thread ..... 678

throw ..... 205

Tight augmentation ..... 790

Top level await ..... 814

Transaction ..... 658

Transform streams ..... 867

Transformation matrix ..... 615

Trap ..... 825

Tree representation ..... 357

True ..... 103

try ..... 202

Two-dimensional array ..... 106

TypeError ..... 203

typeof ..... 233

U

UI ..... 49, 409

UI component ..... 50, 77

UIEvent ..... 435

UML ..... 726

undefined ..... 110

Unidirectional communication ..... 631

Unified modeling language (UML) ..... 726

Uniform resource identifier (URI) ..... 670

Uniform resource locator (URL) ..... 670

Uniform resource name (URN) ..... 670

Unit test ..... 936

unshift() ..... 276

UpperCamelCase notation ..... 93, 239, 243

URI ..... 670

*error* ..... 203

URL ..... 670

URN ..... 670

Use effect ..... 575

User interface ..... 49, 50, 409

*event* ..... 435

**V**

---

Validate form input ..... 475

Validation ..... 49

*native* ..... 475

Value assignment ..... 86

Value range of numbers ..... 97

var ..... 85

Variable ..... 85

*define* ..... 85

Variable declaration ..... 85

Variable initialization ..... 86

Variable name

*allowed characters* ..... 91

*already assigned* ..... 90

*case sensitivity* ..... 92

*meaningful names* ..... 93

*valid* ..... 88

Variant subtag ..... 710

VCS

*centralized* ..... 949

*decentralized* ..... 949

Version control system (VCS) ..... 949

Version management ..... 949

Version management system ..... 949

Visual Studio Code ..... 62

**W**

---

WeakMap ..... 319

WeakSet ..... 324

Wearables ..... 52

Web Animation API ..... 695

Web application ..... 47

Web Hypertext Application Technology

    Working Group ..... 630

Web of Things ..... 52, 909

Web page ..... 47

*print* ..... 496

*search for text* ..... 496

Web Socket API ..... 631, 633

Web Speech API ..... 691

Web Storage API ..... 647

Web view ..... 880

Web Worker API ..... 679

WebGL ..... 600

WebGLRenderingContext ..... 601

Website ..... 47

WebStorm ..... 62

While loop ..... 155

Widget ..... 518, 883

window ..... 487

Windows Mobile ..... 51

World Wide Web Consortium (W3C) ... 362, 630

Wrapper object ..... 329

writable ..... 247

Writable streams ..... 866, 869

**X**

---

XML ..... 518, 519

*parse* ..... 522

*serialize* ..... 523

XML object, convert to strings ..... 523

XML schema ..... 519

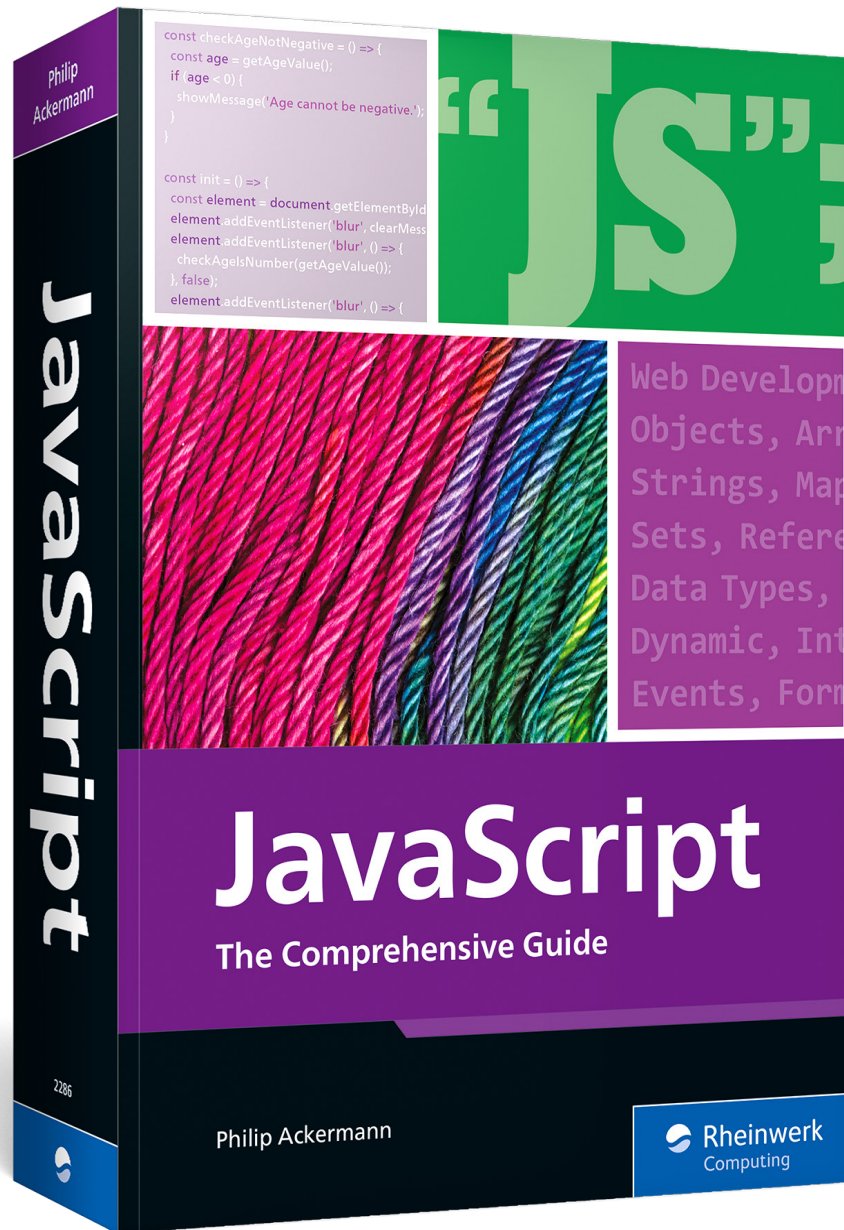
XMLHttpRequest ..... 529

XMLSerializer ..... 523

**Y**

---

yield ..... 821



**Philip Ackermann** is CTO of Cedalo GmbH and author of several reference books and technical articles on Java, JavaScript and web development. His focus is on the design and development of Node.js projects in the areas of Industry 4.0 and Internet of Things.

Philip Ackermann

## JavaScript: The Comprehensive Guide

982 pages, 2022, \$59.95  
ISBN 978-1-4932-2286-5

 [www.rheinwerk-computing.com/5554](https://www.rheinwerk-computing.com/5554)

*We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.*