Preface

JavaScript is an amazing multi-functional language that is used a lot for web development (among other things). Any interaction that happens on web pages is JavaScript in action. In fact, all modern browsers understand JavaScript—and soon you will understand it too.

This book deals with everything you need to know to create JavaScript applications and use JavaScript on web pages. By the time you finish this book, you'll be capable of creating interactive web pages, dynamic applications, and a lot more as you progress on your professional JavaScript journey!

Who this book is for

To get started with this book, you don't need any JavaScript experience. However, if you do have some coding experience, you're likely to go through the book and exercises with a bit more ease. Basic familiarity with HTML and CSS would be of benefit. If you're a first-time programmer, we are honored to welcome you to the world of programming in this book. It may seem difficult in the beginning, but we'll guide you right through it.

What this book covers

Chapter 1, *Getting Started with JavaScript*, covers some fundamentals of the JavaScript language that you'll have to know to understand the rest of the book.

Chapter 2, *JavaScript Essentials*, deals with essentials such as variables, data types, and operators.

Chapter 3, *JavaScript Multiple Values*, covers how to store multiple values in one variable using arrays and objects.

Chapter 4, *Logic Statements*, is where the real fun starts: we are going to use logic statements to make decisions for us!

Chapter 5, *Loops*, accounts for situations when it is necessary to repeat a block of code, which is what we use loops for. We are using different types of loops, such as the `for` and the `while` loop.

Chapter 6, *Functions*, introduces a very useful block for repeating code snippets: functions! This enables us to invoke a specified code block at any time in our script to do something for us. This will help you to not repeat yourself, which is one of the fundamental principles of writing clean code.

Chapter 7, *Classes*, continues with building blocks of JavaScript that help us to structure our application better. We have already seen how to create objects, and with classes we learn how to create a template for objects that we can reuse anytime we need that particular type of object.

Chapter 8, *Built-In JavaScript Methods*, deals with some great built-in functionality. Functions are something we can write ourselves, but we'll find ourselves using the built-in JavaScript functions often whenever we need to do common tasks, such as checking whether something is a number or not.

Chapter 9, *The Document Object Model*, dives into the browser object model and document object model (DOM). This is going to enrich the way we can use JavaScript by a lot. We'll learn what the DOM is, and how we can affect it with JavaScript and change our websites by doing so.

Chapter 10, *Dynamic Element Manipulation Using the DOM*, demonstrates how to manipulate the elements of the DOM dynamically, which will enable you to create modern user experiences. We can change our website as a response to user behavior such as clicking on a button.

Chapter 11, *Interactive Content and Event Listeners*, takes our responses to the user to the next level. For example, we are going to learn how to respond to events such as the cursor leaving an input box and the mouse of the user moving.

Chapter 12, *Intermediate JavaScript*, deals with topics that you'll need to write intermediate JavaScript code, such as regular expressions, recursion, and debugging, to boost the performance of your code.

Chapter 13, *Concurrency*, introduces the topic of concurrency and asynchronous programming, which will allow our code to do multiple things at the same time and be truly flexible.

Chapter 14, *HTML5, Canvas, and JavaScript*, focuses on HTML5 and JavaScript. We'll have seen a lot of both HTML and JavaScript in the previous chapters, but here we'll be focusing on the HTML5-specific features, such as the canvas element.

Chapter 15, *Next Steps*, explores the next steps you could take after you've gotten all the fundamental features of JavaScript down and you are able to write nifty programs using JavaScript. We'll take a look at some of the famous JavaScript libraries and development frameworks, such as Angular, React, and Vue, and we'll have a look at Node.js to see how the backend can be written in JavaScript.

To get the most out of this book

Previous coding experience will help, but is definitely not required. If you have a computer with a text editor (such as Notepad or TextEdit, not Word!) and a browser, you can get started with this book. We encourage you to engage with the exercises and projects, and experiment continually while you go through the chapters, to ensure you are comfortable with each concept before moving on.

Download the example code files

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/JavaScript-from-Beginner-to-Professional`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781800562523_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. For example; "We also need to let the browser know what kind of document we're working on with the `<!DOCTYPE>` declaration."

A block of code is set as follows:

```
<html>
  <script type="text/javascript">
    alert("Hi there!");
  </script>
</html>
```

Any command-line input or output is written as follows:

```
console.log("Hello world!")
```

Bold: Indicates a new term, an important word, or words that you see on the screen, for example, in menus or dialog boxes, also appear in the text like this. For example: "If you right-click and select **Inspect** on macOS systems, you will see a screen appear, similar to the one in the following screenshot."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com`, and mention the book's title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book we would be grateful if you would report this to us. Please visit, `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `http://authors.packtpub.com`.

Share Your Thoughts

Once you've read *JavaScript from Beginner to Professional*, we'd love to hear your thoughts! Please `click here to go straight to the Amazon review page` for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

1

Getting Started with JavaScript

It appears you have decided to start learning JavaScript. Excellent choice! JavaScript is a programming language that can be used on both the server side and client side of applications. The server side of an application is the backend logic that usually runs on computers in data centers and interacts with the database, while the client side is what runs on the device of the user, often the browser for JavaScript.

It is not unlikely that you have used functionality written in JavaScript. If you have used a web browser, such as Chrome, Firefox, Safari, or Edge, then you definitely have. JavaScript is all over the web. If you enter a web page and it asks you to accept cookies and you click OK, the popup disappears. This is JavaScript in action. And if you want to navigate a website and a sub-menu opens up, that means more JavaScript. Often, when you filter products in a web shop, this involves JavaScript. And what about these chats that start talking to you after you have been on a website for a certain number of seconds? Well, you guessed it—JavaScript!

Pretty much any interaction we have with web pages is because of JavaScript; the buttons you are clicking, birthday cards you are creating, and calculations you are doing. Anything that requires more than a static web page needs JavaScript.

In this chapter, we will cover the following topics:

- Why should you learn JavaScript?
- Setting up your environment
- How does the browser understand JavaScript?

- Using the browser console
- Adding JavaScript to a web page
- Writing JavaScript code

Note: exercise, project and self-check quiz answers can be found in the *Appendix*.

Why should you learn JavaScript?

There are many reasons why you should want to learn JavaScript. JavaScript originates from 1995, and is often considered the most widely used programming language. This is because JavaScript is the language that web browsers support and understand. You have everything you need to interpret it already installed on your computer if you have a web browser and text editor. There are better setups, however, and we will discuss these later in this chapter.

It is a great programming language for beginners, and most advanced software developers will know at least some JavaScript because they will have run into it at some point. JavaScript is a great choice for beginners for a number of reasons. The first reason is that you can start building really cool apps using JavaScript sooner than you could imagine. By the time you get to *Chapter 5*, *Loops*, you will be able to write quite complex scripts that interact with users. And by the end of the book, you will be able to write dynamic web pages to do all sorts of things.

JavaScript can be used to write many different types of applications and scripts. It can be used for programming for the web browser, but also the logic layer of code that we cannot see (such as communication with the database) of an application can be programmed in JavaScript, along with games, automation scripts, and a plethora of other purposes. JavaScript can also be used for different programming styles, by which we mean ways to structure and write code. How you would go about this depends on the purpose of your script. If you've never coded before, you may not quite grasp these concepts, and it's not entirely necessary to at this stage, but JavaScript can be used for (semi) object-oriented, functional, and procedural programming, which are just different programming paradigms.

There are a ton of libraries and frameworks you can use once you get the basics of JavaScript down. These libraries and frameworks will really enhance your software life and make it a lot easier and possible to get more done in less time. Examples of these great libraries and frameworks include React, Vue.js, jQuery, Angular, and Node.js. Don't worry about these for now; just see them as a bonus for later. We will cover some of them briefly at the very end of this book.

Finally, we'll mention the JavaScript community. JavaScript is a very popular programming language, and many people are using it. As a beginner in particular, there won't be a problem for which you cannot find a solution on the internet.

The community of JavaScript is huge. The popular Stack Overflow forum contains lots of help for all sorts of coding issues and has an enormous section on JavaScript. You'll find yourself running into this web page a lot while googling problems and tips and tricks.

If JavaScript is your first programming language, you are new to the whole software community and you are in for a treat. Software developers, no matter the language, love to help one another. There are forums and tutorials online and you can find answers to almost all your questions. As a beginner, it can be hard to understand all the answers though. Just hang in there, keep trying and learning, and you will understand it soon enough.

Setting up your environment

There are many ways in which you can set up a JavaScript coding environment. For starters, your computer probably already has all the minimal things you will need to code JavaScript. We recommend you make your life a little bit easier and use an IDE.

Integrated Development Environment

An **Integrated Development Environment** (**IDE**) is a special application that is used to write, run, and debug code. You can just open it like you would any program. For example, to write a text document, you need to open the program, select the right file, and start to write. Coding is similar. You open the IDE and write the code. If you want to execute the code, the IDE often has a special button for this. Pressing this button will run the code from inside the IDE. For JavaScript, you might find yourself opening your browser manually in certain cases though.

An IDE does do more than that though; it usually has syntax highlighting. This means that certain elements in your code will have a certain color, and you can easily see when something is going wrong. Another great feature is the autosuggest feature, where the editor helps you with the options you have at the place where you are coding. This is often called code completion. Many IDEs have special plugins so you can make working with other tools more intuitive and add features to it, for example, a hot reload in the browser.

There are many IDEs out there and they differ in what they have to offer. We use Visual Studio Code throughout the book, but that's a personal preference. Other popular ones at the time of writing include Atom, Sublime Text, and WebStorm.

There are many IDEs and they keep on appearing, so chances are the most popular one at the time you are reading is not on this list. There are many other options. You can do a quick search on the web for JavaScript IDEs. There are a few things to pay attention to when selecting an IDE. Make sure that it supports syntax highlighting, debugging, and code completion for JavaScript.

Web browser

You will also need a web browser. Most browsers are perfectly fine for this, but it's better not to use Internet Explorer, which doesn't support the latest JavaScript features. Two good options would be Chrome and Firefox. They support the latest JavaScript features and helpful plugins are available.

Extra tools

There are many extra things you can use while coding, for example, browser plugins that will help you to debug or make things easier to look at. You don't really need any of them at this point, but keep an open mind whenever you come across a tool that others are very excited about.

Online editor

It may be the case that you don't have access to a computer, perhaps just a tablet, or that you cannot install anything on your laptop. There are great online editors out there for these scenarios as well. We don't name any, since they are evolving rapidly and probably will be old by the time you are reading this. But if you do a web search for `online JavaScript IDE`, you will find plenty of online options where you can just start coding JavaScript and hit a button to run it.

How does the browser understand JavaScript?

JavaScript is an interpreted language, which means that the computer understands it while running it. Some languages get processed before running, this is called compiling, but not JavaScript. The computer can just interpret JavaScript on the fly. The "engine" that understands JavaScript will be called the interpreter here.

A web page isn't just JavaScript. Web pages are written in three languages: HTML, CSS, and JavaScript.

HTML determines what is on the page; the content of the page is in there. If there is a paragraph on the page, the HTML of the page contains a paragraph. And if there is a heading, HTML was used to add a heading, and so forth. HTML consists of elements, also called tags. They specify what is on the page. Here is a little sample that will create a web page with the text `Hello world` on it:

```
<html>
  <body>
    Hello world!
  </body>
</html>
```

In *Chapter 9*, *The Document Object Model*, we have a little crash course in HTML, so don't worry if you have never seen it.

CSS is the layout of the web page. So for example, if the text color is blue, this is done by CSS. Font size, font family, and position on the page are all determined by CSS. JavaScript is the final piece in the puzzle, which defines what the web page can do and how it can interact with the user or the backend.

When dealing with JavaScript, you will come across the term **ECMAScript** sooner or later. This is the specification or standardization for the JavaScript language. The current standard is **ECMAScript 6** (also referred to as **ES6**). Browsers use this specification to support JavaScript (in addition to some other topics such as **Document Object Model** (**DOM**), which we'll see later). JavaScript has many implementations that might differ slightly, but ECMAScript can be considered the basic specification that the JavaScript implementation will definitely include.

Using the browser console

You may have seen this already, or not, but web browsers have a built-in option to see the code that makes the web page you are on possible. If you hit *F12* on a Windows computer while you are in the web browser, or you right-click and select **Inspect** on macOS systems, you will see a screen appear, similar to the one in the following screenshot.

It might work slightly differently on your browser on your machine, but right-clicking and selecting **Inspect** generally does the trick:

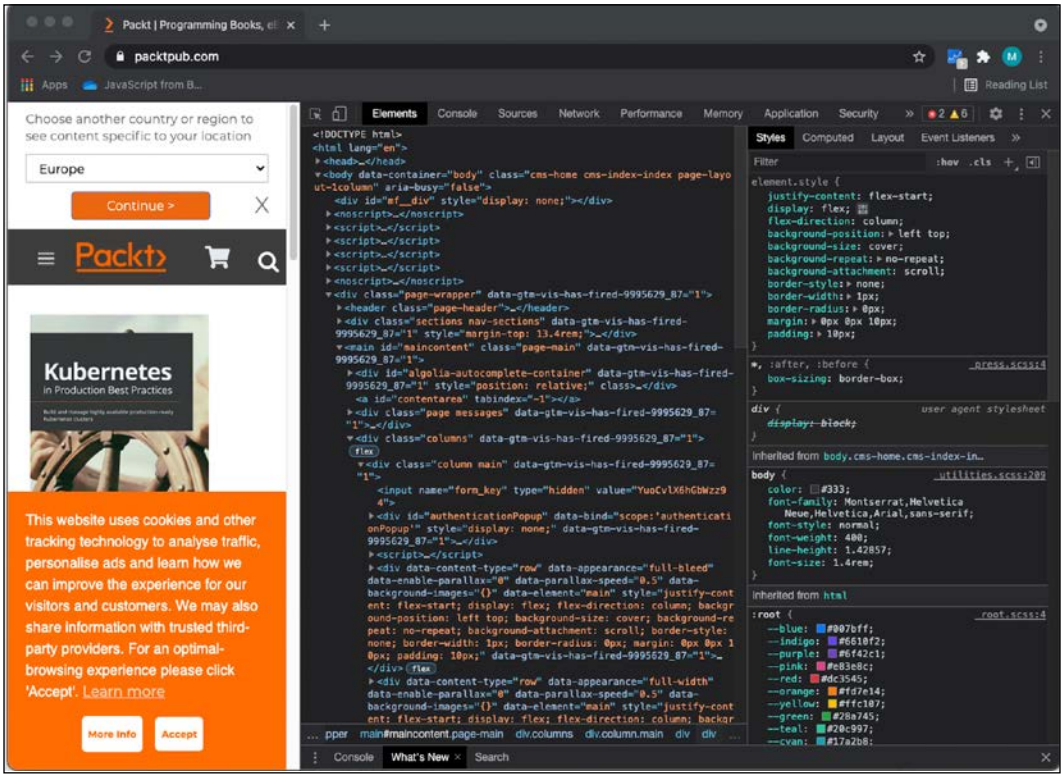

Figure 1.1: Browser console on the Packt website

This screenshot contains multiple tabs at the top. We are now looking at the element tabs, which contain all the HTML and CSS (remember those?). If you click on the console tab, you will find at the bottom of the panel a place where you can insert some code directly. You may see some warnings or error messages in this tab. This is not uncommon, and don't worry about it if the page is working.

The console is used by developers to log what is going on and do any debugging. Debugging is finding the problem when an application is not displaying the desired behavior. The console gives some insights as to what is happening if you log sensible messages. This is actually the first command we are going to learn:

```
console.log("Hello world!");
```

If you click on this console tab, enter the first JavaScript code above, and then hit *Enter*, this will show you the output of your code therein. It will look like the following screenshot:

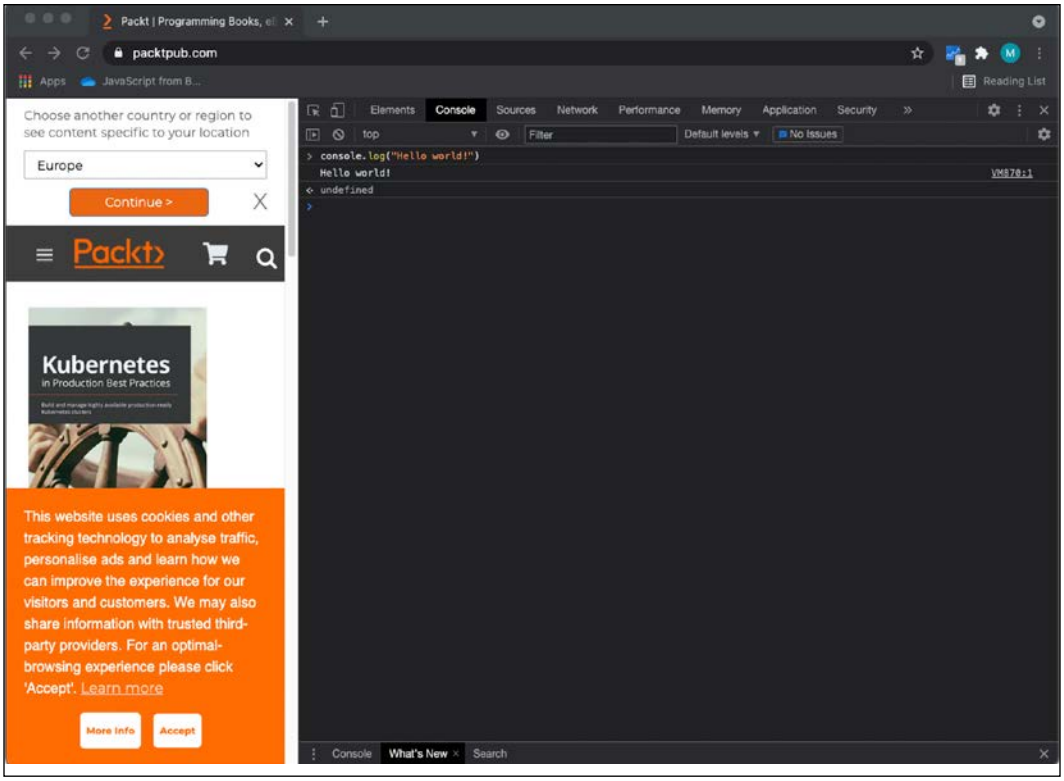

Figure 1.2: JavaScript in the browser console

You will be working with the `console.log()` statement a lot throughout the book in your code to test your code snippets and see the results. There are also other console methods, such as `console.table()`, that create a table when the inputted data can be presented as a table. Another console method is `console.error()`, which will log the inputted data, but with a styling that draws attention to the fact that it's an error.

Practice exercise 1.1

Working with the console:

1. Open the browser console, type `4 + 10`, and press *Enter*. What do you see as the response?
2. Use the `console.log()` syntax, placing a value within the rounded brackets. Try entering your name with quotes around it (this is to indicate the fact that it's a text string—we'll get to this in the next chapter).

Adding JavaScript to a web page

There are two ways to link JavaScript to a web page. The first way is to type the JavaScript directly in the HTML between two `<script>` tags. In HTML, the first tag, `<script>`, is to declare that the following script will be executed. Then we have the content that should be inside this element. Next, we close the script with the same tag, but preceded by a forward slash, `</script>`. Or you can link a JavaScript file to the HTML file using the script tag at the head of the HTML page.

Directly in HTML

Here is an example of how to write a very simple web page that will give a pop-up box saying `Hi there!`:

```
<html>
  <script type="text/javascript">
    alert("Hi there!");
  </script>
</html>
```

If you store this as a `.html` file, and open the file in your browser, you will get something like the following screenshot. We will be storing this one as `Hi.html`:

Figure 1.3: JavaScript made this popup with the text "Hi there!" appear

The `alert` command will create a popup that provides a message. This message is specified between the parentheses behind the alert.

Right now, we have the content directly within our `<html>` tags. This is not a best practice. We will need to create two elements inside `<html>`—`<head>` and `<body>`. In the head element, we write metadata and we also use this part later to connect external files to our HTML file. In the body, we have the content of the web page.

We also need to let the browser know what kind of document we're working on with the `<!DOCTYPE>` declaration. Since we're writing JavaScript inside an HTML file, we need to use `<!DOCTYPE html>`. Here's an example:

```
<!DOCTYPE html>
<html>

<head>
  <title>This goes in the tab of your browser</title>
</head>

<body>
The content of the webpage
  <script>
    console.log("Hi there!");
  </script>
</body>

</html>
```

This example web page will display the following: `The content of the webpage`. If you look in the browser console, you'll find a surprise! It has executed the JavaScript as well and logs `Hi there!` in the console.

Practice exercise 1.2

JavaScript in an HTML page:

1. Open your code editor and create an HTML file.
2. Within your HTML file, set up the HTML tags, doctype, HTML, head, and body, and then proceed and add the script tags.
3. Place some JavaScript code within the script tags. You can use `console.log("hello world!")`.

Linking an external file to our web page

You could also link an external file to the HTML file. This is considered a better practice, as it organizes code better and you can avoid very lengthy HTML pages due to the JavaScript. In addition to these benefits, you can reuse the JavaScript on other web pages of your website without having to copy and paste. Say that you have the same JavaScript on 10 pages and you need to make a change to the script. You would only have to change one file if you did it in the way we are showing you in this example.

First, we are going to create a separate JavaScript file. These files have the postfix `.js`. I'm going to call it `ch1_alert.js`. This will be the content of our file:

```
alert("Saying hi from a different file!");
```

Then we are going to create a separate HTML file (using the postfix `.html` again). And we are going to give it this content:

```
<html>
  <script type="text/javascript" src="ch1_alert.js"></script>
</html>
```

Make sure that you put the files in the same location, or that you specify the path to the JavaScript file in your HTML. The names are case-sensitive and should match exactly.

You have two options. You can use a relative path and an absolute path. Let's cover the latter first since that is the easiest to explain. Your computer has a root. For Linux and macOS, it is `/`, and for Windows, it is often `C:/`. The path to the file starting from the root is the absolute path. This is the easiest to add because it will work on your machine. But there is a catch: on your machine, if this website folder later gets moved to a server, the absolute path will no longer work.

The second, safer option is relative paths. You specify how to get there from the file you are in at that time. So if it's in the same folder, you will only have to insert the name. If it's in a folder called "example" that is inside the folder that your file is in, you will have to specify `example/nameOfTheFile.js`. And if it's a folder up, you would have to specify `../nameOfTheFile.js`.

If you open the HTML file, this is what you should get:

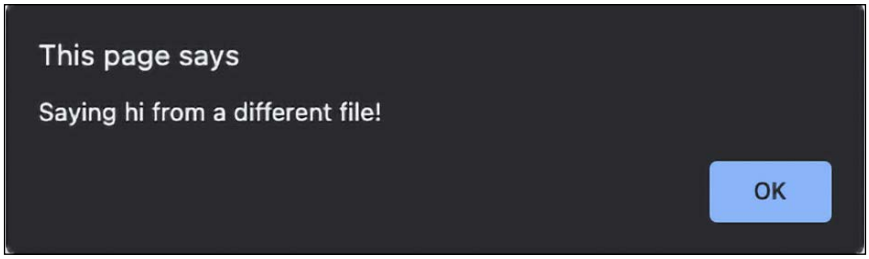

Figure 1.4: Popup created by JavaScript in a different file

Practice exercise 1.3

Linking to a JS JavaScript file:

1. Create a separate file called `app` with the extension `.js`.
2. Within the `.js` file, add some JavaScript code.
3. Link to the separate `.js` file within the HTML file you created in *Practice exercise 1.2*.
4. Open the HTML file within your browser and check to see whether the JavaScript code ran properly.

Writing JavaScript code

So, we now have lots of context, but how do you actually write JavaScript code? There are some important things to keep in mind, such as how to format the code, using the right indentation level, using semicolons, and adding comments. Let's start with formatting code.

Formatting code

Code needs to be formatted well. If you have a long file with many lines of code and you didn't stick to a few basic formatting rules, it is going to be hard to understand what you've written. So, what are the basic formatting rules? The two most important for now are indentations and semicolons. There are also naming conventions, but these will be addressed for every topic that is yet to come.

Indentations and whitespace

When you are writing code, often a line of code belongs to a certain code block (code between two curly brackets `{ like this }`) or parent statement. If that is the case, you give the code in that block one indentation to make sure that you can see easily what is part of the block and when a new block starts. You don't need to understand the following code snippet, but it will demonstrate readability with and without indentations.

Without new lines:

```
let status = "new"; let scared = true; if (status === "new") { console.
log("Welcome to JavaScript!");  if (scared) { console.log("Don't worry
you will be fine!"); } else { console.log("You're brave! You are going
to do great!"); } } else { console.log("Welcome back, I knew you'd like
it!"); }
```

With new lines but without indentation:

```
let status = "new";
let scared = true;
if (status === "new") {
console.log("Welcome to JavaScript!");
if (scared) {
console.log("Don't worry you will be fine!");
} else {
console.log("You're brave! You are going to do great!");
}
} else {
console.log("Welcome back, I knew you'd like it!");
}
```

With new lines and indentation:

```
let status = "new";
let scared = true;
if (status === "new") {
  console.log("Welcome to JavaScript!");
  if (scared) {
    console.log("Don't worry you will be fine!");
  } else {
    console.log("You're brave! You are going to do great!");
  }
} else {
  console.log("Welcome back, I knew you'd like it!");
}
```

As you can see, you can now easily see when the code blocks end. This is where the `if` has a corresponding `}` at the same indentation level. In the example without indentations, you would have to count the brackets to determine when the `if` block would end. Even though it is not necessary for working code, make sure to use indentation well. You will thank yourself later.

Semicolons

After every statement, you should insert a semicolon. JavaScript is very forgiving and will understand many situations in which you have forgotten one, but develop the habit of adding one after every line of code early. When you declare a code block, such as an `if` statement or loop, you should not end with a semicolon. It is only for the separate statements.

Code comments

With comments, you can tell the interpreter to ignore some lines of the file. They won't get executed if they are comments. It is often useful to be able to avoid executing a part of the file. This could be for the following reasons:

1. You do not want to execute a piece of code while running the script, so you comment it out so it gets ignored by the interpreter.
2. Metadata. Adding some context to the code, such as the author, and a description of what the file covers.
3. Adding comments to specific parts of the code to explain what is happening or why a certain choice has been made.

There are two ways to write comments. You can either write single-line comments or multi-line comments. Here is an example:

```
// I'm a single line comment
// console.log("single line comment, not logged");

/* I'm a multi-line comment. Whatever is between the slash asterisk and
the asterisk slash will not get executed.
console.log("I'm not logged, because I'm a comment");
*/
```

In the preceding code snippet, you see both commenting styles. The first one is single-line. This can also be an inline comment at the end of the line. Whatever comes after the `//` on the line will get ignored. The second one is multiline; it is written by starting with `/*` and ending with `*/`.

Practice exercise 1.4

Adding comments:

1. Add a new statement to your JavaScript code by setting a variable value. Since we will cover this in the next chapter, you can use the following line:
   ```
   let a = 10;
   ```
2. Add a comment at the end of the statement indicating that you set a value of `10`.
3. Print the value using `console.log()`. Add a comment explaining what this will do.

4. At the end of your JavaScript code, use a multiple-line comment. In a real production script, you might use this space to add a brief outline of the purpose of the file.

Prompt

Another thing we would like to show you here is also a command prompt. It works very much like an alert, but instead, it takes input from the user. We will learn how to store variables very soon, and once you know that, you can store the result of this prompt function and do something with it. Go ahead and change the `alert()` to a `prompt()` in the `Hi.html` file, for example, like this:

```
prompt("Hi! How are you?");
```

Then, go ahead and refresh the HTML. You will get a popup with an input box in which you can enter text, as follows:

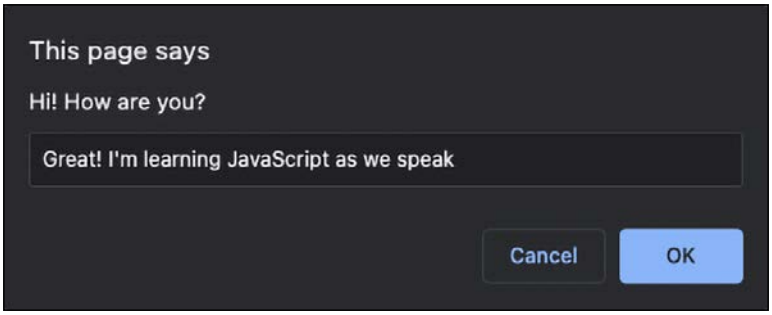

Figure 1.5: Page prompting for use input

The value you (or any other user) enter will be returned to the script, and can be used in your code! This is great for getting user input to shape the way your code works.

Random numbers

For the purpose of fun exercises in the early chapters of this book, we would like you to know how to generate a random number in JavaScript. It is absolutely fine if you don't really understand what is going on just yet; just know that this is the command to create a random number:

```
Math.random();
```

We can do it in the console and see the result appear if we log it:

```
console.log(Math.random());
```

This number will be a decimal between 0 and 1. If we want a number between 0 and 100, we can multiply it by 100, like this:

```
console.log(Math.random() * 100);
```

Don't worry, we will cover mathematic operators in *Chapter 2, JavaScript Essentials*.

If we don't want to have a decimal result, we can use the `Math.floor` function on it, which is rounding it down to the nearest integer:

```
console.log(Math.floor(Math.random() * 100));
```

Don't worry about not getting this yet. This will be explained in more detail further on in the book. In *Chapter 8, Built-In JavaScript Methods*, we will discuss built-in methods in more detail. Until then, just trust us that this does generate a random number between 0 and 100.

Chapter project

Creating an HTML file and a linked JavaScript file

Create an HTML file and create a separate JavaScript file. Then, connect to the JavaScript file from the HTML file.

1. In the JavaScript file, output your name into the console and add a multiple-line comment to your code.
2. Try commenting out the console message in your JavaScript file so that nothing shows in the console.

Self-check quiz

1. What is the HTML syntax to add an external JavaScript file?
2. Can you run JavaScript in a file with a JS extension in your browser?
3. How do you write a multiple-line comment in JavaScript?
4. What is the best way to remove a line of code from running that you might want to keep as you debug?

Summary

Nicely done! You have made a start with JavaScript! In this chapter, we have discussed a lot of context, which you will need to know before starting to code JavaScript. We saw that we can use JavaScript for many purposes, and one of the most popular use cases is the web. Browsers can work with JavaScript because they have a special part, called an interpreter, that can process JavaScript. We saw that we have multiple options for writing JavaScript on our computer. We will need an IDE, a program that we can use to write and run our code.

Adding JavaScript to a web page can be done in several ways. We saw how to include it in the script element and how to add a separate JavaScript file to a page. We ended this chapter with some important general notes on how to write well-structured, readable, and easy-to-maintain code that is well documented with comments. We also saw that we can write to the console with our `console.log()` method and ask for user input using `prompt()`. Lastly, we also saw that we can generate random numbers with the `Math.random()` function.

Next, we'll look at JavaScript's basic data types and the operators that can be used to manipulate them!