This is a pre-release section of the work “Begin to Code with JavaScript”. Everything is subject to change, especially the jokes. You can find the “Begin to Code with JavaScript” podcast page here:

www.robmiles.com/jspodcast

The code samples for the book and links to the screencasts can be found here:

www.begintocodewithjavascript.com

Feel free to send constructive comments to writing@begintocodewithjavascript.com

Introduction

Programming is the most creative thing you can learn how to do. Why? If you learn to paint, you can create pictures. If you learn to play the violin, you can make music. But if you learn to program, you can create entirely new experiences (and you can make pictures and music too, if you wish). Once you've started on the programming path, there's no limit to where you can go. There are always new devices, technologies, and marketplaces where you can use your programming skills.

Think of this book as your first step on a journey to programming enlightenment. The best journeys are undertaken with a destination in mind, and the destination of this journey is "usefulness." By the end of this book, you will have the skills and knowledge to write useful programs and make them available to anyone in the world.

But first, a word of warning. I would not say that learning to write programs is easy. This is for two reasons:

- If I tell you it's easy, and you still can't do it you might feel bad about this (and rather cross with me)
- If I tell it's is easy and you manage to do it, you might think that it isn't worth doing.

Learning to program is not easy. It's a kind of difficult that you might not have seen before. Programming is all about detail and sequencing. You must learn how the computer does things and how to express what you want it to do.

Imagine that you were lucky enough to be able to afford your own personal chef. At the start you would have to explain things like "If it is sunny outside I like orange juice and a grapefruit for breakfast, but if it is raining I'd like a bowl of porridge and a big mug of coffee". Occasionally your chef would make mistakes, perhaps you would get a black coffee rather than the latte that you wanted. However, over time you would add more detail to your instructions until your chef knew exactly what to do.

A computer is like a chef that doesn't even know how to cook. Rather than saying "make me a coffee" you would have to say, "Take the brown powder from the coffee bag and add it to hot water". Then you would have to explain how to make hot water, and how you must be careful with the kettle and so on. This is hard work.

It turns out that the key to success as a programmer is much the same as for many other endeavors. To become a world-renowned violin player, you will have to practice a lot. The same is true for programming. You must spend a lot of time working on your programs to acquire code-writing skills. However, the good news is that just as a violin player really enjoys making the instrument sing, making a computer do exactly what you want turns out to be a very rewarding experience. It gets even more enjoyable when you see other people using programs that you've written and finding them useful and fun to use.

How this book fits together

I've organized this book in three parts. Each part builds on the previous one with the aim of turning you into a successful programmer. We start off discovering the environment in which JavaScript programs run. Then we learn the fundamentals of programming and we finish by making some properly useful (and fun) programs.

Part 1: The JavaScript world

The first part gets you started. You'll discover the environment in which JavaScript programs run and learn how to create web pages containing JavaScript programs.

Part 2: Coding with JavaScript

Part 2 describes the features of the JavaScript that you use to create programs that work on data. You will pick up some fundamental programming skills that apply to a wide range of other languages, and get you thinking about what it is that programs actually do. You'll find out you how to break large programs into smaller elements and how you can create custom data types that reflect the specific problem being solved.

Part 3: Useful JavaScript

Now that you can make JavaScript programs it's time to have some fun with them. You'll discover how to create good looking applications, how to make programs that are secure and reliable and finish off with a bit of game development.

How you will learn

In each chapter, I will tell you a bit more about programming. I'll show you how to do something, and then I'll invite you to make something of your own by using what you've learned. You'll never be more than a page or so away from doing something or making something unique and personal. After that, it's up to you to make something amazing!

You can read the book straight through if you like, but you'll learn much more if you slow down and work with the practical parts along the way. Like learning to ride a bicycle, you'll learn by *doing*. You must put in the time and practice to learn how to do it. But this book will give you the knowledge and confidence to try your hand at programming, and it will also be around to help you if your programming doesn't turn out as you expected. Here are some elements in the book that will help you learn by doing:

Make Something Happen

Yes, the best way to learn things is by doing, so you'll find "Make Something Happen" elements throughout the text. These elements offer ways for you to practice your programming skills. Each starts with an example and then introduces some steps you can try on your own. Everything you create will run on Windows, macOS, or Linux.

Code Analysis

A great way to learn how to program is by looking at code written by others and working out what it does (and sometimes why it doesn't do what it should). The book contains over 150 sample programs for you go look at. In this book's "Code Analysis" challenges, you'll use your deductive skills to figure out the behavior of a program, fix bugs, and suggest improvements.

What Could Go Wrong?

If you don't already know that programs can fail, you'll learn this hard lesson soon after you begin writing your first program. To help you deal with this in advance, I've included "What Could Go Wrong?" elements, which anticipate problems you might have and provide solutions to those problems. For example, when I introduce something new, I'll sometimes spend some time considering how it can fail and what you need to worry about when you use the new feature.

Programmer's Points

I've spent a lot of time teaching programming. But I've also written many programs and sold a few to paying customers. I've learned some things the hard way that I really wish I'd known at the start. The aim of "Programmer's Points" is to give you this information up front so that you can start taking a professional view of software development as you learn how to do it.

"Programmer's Points" cover a wide range of issues, from programming to people to philosophy. I strongly advise you to read and absorb these points carefully—they can save you a lot of time in the future!

What you will need

You'll need a computer and some software to work with the programs in this book. I'm afraid I can't provide you with a computer, but in the first chapter you'll find out how you can get started with nothing more than a computer and a web browser. Later you'll discover how to use the Visual Studio Code editor to create JavaScript programs.

Using a PC or laptop

You can use Windows, macOS, or Linux to create and run the programs in the text. Your PC doesn't have to be particularly powerful, but these are the minimum specifications I'd recommend:

- A 1 GHz or faster processor, preferably an Intel i5 or better.

- At least 4 gigabytes (GB) of memory (RAM), but preferably 8 GB or more.
- 256 GB hard drive space. (The JavaScript frameworks and Visual Studio Code installations take about 1 GB of hard drive space.)

There are no specific requirements for the graphics display, although a higher-resolution screen will enable you to see more when writing your programs.

Using a mobile device

You can run JavaScript programs on a mobile phone or tablet by visiting the web pages in which the programs are held. There are also some applications that can be used to create and run JavaScript programs but my experience has been that a laptop or desktop computer is a better place to work.

Using a Raspberry Pi

If you want to get started in the most inexpensive way possible you can use a Raspberry Pi running the Raspbian Operating System. This has a Chromium compatible browser and is also capable of running Visual Studio Code.

Sample Code

In every chapter in this book, I'll demonstrate and explain programs that teach you how to begin to program—and that you can then use to create programs of your own. You can download this book's sample code from GitHub by following the link here:

http://www.begintocodewithjavascript.com/code

GitHub was developed as a software development platform but it has turned out to be much more than that. It is a place where anyone can store files and share them. All the sample programs used in the book are held on GitHub.

At the start of the book you'll discover how to use GitHub to make your own copy of the sample programs. You can then use GitHub to publish JavaScript enabled web pages for anyone in the world to view.

You will need to connect to the internet and create a GitHub account (it is free) to do this.

Electronic media

For the important content elements, I've made some videos. The book text will contain screenshots that you can use, but these can go out of date. Follow the links to the walkthroughs to get the latest steps to follow. There are also some audio recordings you can listen to if you feel brave. You can find all these here:

http://www.begintocodewithjavascript.com/media

Acknowledgments

Thanks to everyone for giving me a chance to do it all again.

1 The world of JavaScript

We are going to start our journey by looking at the world of JavaScript. We'll begin by considering just what it is that a programming language does. Then we'll investigate the JavaScript programming language and discover how JavaScript programs get to run on your computer. We'll learn how web pages provide an environment for JavaScript and how to use HyperText Markup Language (HTML) and Cascading StyleSheets (CSS) to create containers for our JavaScript programs. We'll discover just how powerful modern web browsers are as software development tools and how to have a conversation with JavaScript from within a browser. We'll also learn how to manage our software source code and share it with others.

1

Running JavaScript

What you will learn

Programmers have a set of tools and techniques they use when they create programs. In this chapter, you're going to discover how JavaScript programs run on a computer. You'll also have your first of many conversations with the JavaScript command prompt and investigate your first JavaScript program. Finally, you'll download the Git and Visual Studio Code tools and the example programs for this book and do some simple editing.

What is JavaScript?

Before we go off and look at some JavaScript it's worth considering just what we are running. JavaScript is a *programming language*. In other words, it's a language that you use to write programs. A program is a set of instructions that tells a computer how to do something. We can't use a "proper" language like English to do this because "proper English" is just too confusing for a computer to understand. As an example, I give you the doctor's instructions:

```
"Drink your medicine after a hot bath."
```

We would probably have a hot bath and then drink our medicine. A computer would probably drink the hot bath and then drink its medicine. You can interpret the above instructions either way because the English language allows you to write ambiguous statements. Programming languages must be designed so that instructions written using them are not open to interpretation, they must tell the computer precisely what to do. This usually means breaking actions down into a sequence of simple steps:

```
Step1: Take a hot bath
Step2: Drink your medicine
```

We can get this effect in English (as you can see above) but a programming language forces us to write instructions in this way. JavaScript is one of many programming languages which have been invented to provide humans with a way of telling the computer what to do.

In my programming career, I've learnt many different languages over the years and I confidently expect to have to learn even more in the future. None of them are perfect, and I see each of them as a tool that I would use in a particular situation, just like I would choose a different tool depending on whether I was making a hole in a brick wall, a pane of glass or a piece of wood.

Some people get very excited when talking about the "best" programming language. I'm quite happy to discuss what makes the best programming languages, just like I'm happy to tell you all about my favorite type of car, but I don't see this as something to get worked up about. I love JavaScript for its power and the ease with which I can distribute my code. I love Python for its expressiveness and how I can create complex solutions with tiny bits of code. I love the C# programming language for the way it pushes me to produce well-structured solutions. I love the C++ programming language for the way that it gives me absolute control of hardware underneath my program. And so on. JavaScript does have things about it which make me want to tear my hair out in frustration. But that's true of the other languages too. And all programming languages have things about them I love. But most of all I love JavaScript for the way that I can use it to pay my bills.

Programmer's Point

The best programming language for you is the one that pays you the most

I think it is very fitting that the first programmer's point is one that has a strong commercial focus. Whenever I get asked which is the "best" programming language I always say that my favorite language is the one that I get paid the most to use. It turns out that I'll write in any programming language if the price is right.

I strongly believe that you can enjoy programming well in any language, and that includes JavaScript. Conversely, you can have a horrible time writing bad programs in any language. The language is just the medium which you use to express your ideas.

So, if you tell someone that you're writing JavaScript programs and they tell you that it's not a very good programming language for reasons that you don't understand, just show them how many jobs there are out there for people who can write JavaScript code.

JavaScript origins

You might think that programming languages are a bit like space rockets, in that they are designed by white-coated scientists with mega-brains who get everything right first time and always produce perfect solutions. However, this is not the case. Programming languages have been developed over the years for all kinds of reasons, including ones like “it seemed a good idea at the time”.

JavaScript was invented by Brendan Eich of Netscape Communications Corporation and first appeared in a Netscape web browser that was released at the end of 1995. The language had a variety of names before the company decided on JavaScript. It turned out to be a poor choice of name because it makes it easy to confuse JavaScript with the Java programming language, which is actually quite different from JavaScript.

JavaScript was intended as a simple way of making web pages interactive. Its name reflects the way that it was supposed to be used alongside Java applications (called *applets*) running in a web browser. However, JavaScript was extended beyond all the expectations of its creator and is now one of the most popular programming languages in the world. Whenever you visit a web site it is almost certain that you will be talking to a JavaScript program.

This book will teach you JavaScript, but actually I’m trying to turn you into a programmer. The fundamentals of program creation are the same for JavaScript and pretty much all programming languages. Once you’ve learned how to write JavaScript you’ll be able to transfer this skill into many other languages, including C++, C#, Visual Basic and Python. It’s a bit like the way that once you have learned to drive you can drive any vehicle. When you are using a strange car you just need to find out where the various switches and controls are, and then you can set off on your journey.

JavaScript and the web browser

The inventor of JavaScript intended for it to be used in a web browser and that is where we are going to start using it. It is possible to create JavaScript programs that run outside the browser, we will consider how to do this in the third part of this text. You can use any modern browser, but the exercises in this text use a browser based on the Chromium framework. I’m using Microsoft Edge Chromium which is available for Windows PC and Apple Macintosh. You can use Google Chrome or the Chromium browser for Linux if you prefer.

Our first brush with JavaScript

You’ve reached a significant point in the process of learning how to program. You’re about to begin exploring how programs work. This is a bit like opening the front door of a new apartment or house or getting into a shiny new car you’ve bought. It’s an exciting time, so take a deep breath, find a nice cup (or glass) of something you like to drink and settle down comfortably.

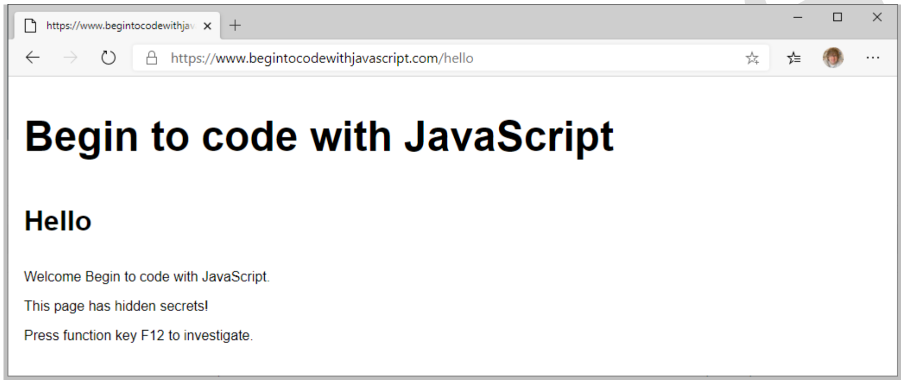
You are going to start by doing something that you’ve done thousands of times in the past. You are going to visit a site on the World Wide Web. But then, with a single press of a key, you're going to explore a world behind the web page and get a glimpse of the role that JavaScript plays in making it work.

Make Something Happen

A web page with secrets

First you need to open your browser. Then visit the web page:

http://www.begintocodewithjavascript.com/hello



https://www.begintocodewithjavascript.com/hello

Begin to code with JavaScript

Hello

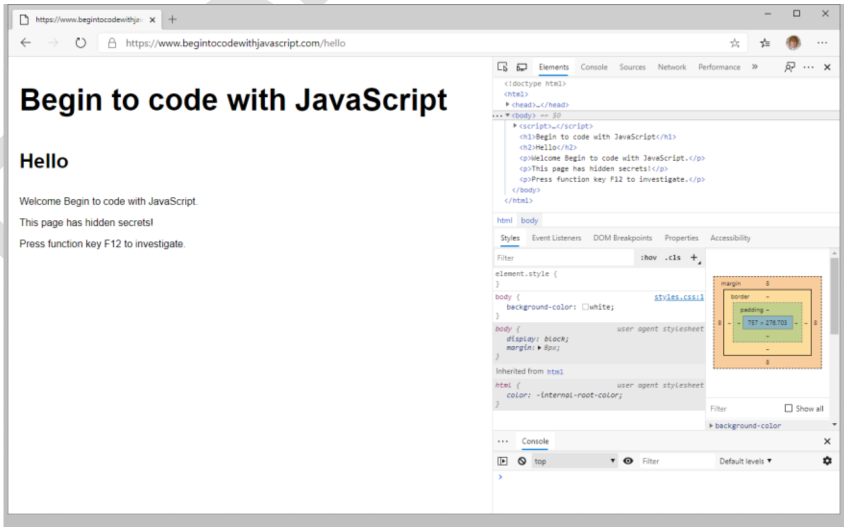
Welcome Begin to code with JavaScript.

This page has hidden secrets!

Press function key F12 to investigate.

1.1 Ch01_inset01_01 A web page with secrets

This looks like a very ordinary web page. But it holds a secret behavior that you can find by pressing the F12 key on your keyboard.



https://www.begintocodewithjavascript.com/hello

Begin to code with JavaScript

Hello

Welcome Begin to code with JavaScript.

This page has hidden secrets!

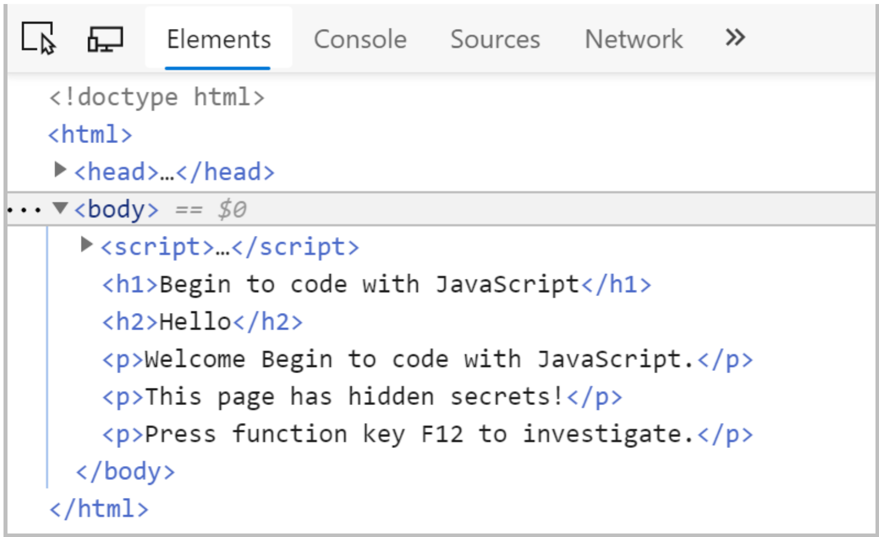
Press function key F12 to investigate.

```
<!doctype html>
<html>
 ▶<head>…</head>
…▼<body> == $0
   ▶<script>…</script>
    <h1>Begin to code with JavaScript</h1>
    <h2>Hello</h2>
    <p>Welcome Begin to code with JavaScript.</p>
    <p>This page has hidden secrets!</p>
    <p>Press function key F12 to investigate.</p>
  </body>
</html>
```

```
element.style {
}
body {                                   styles.css:1
   background-color: white;
}
body {                          user agent stylesheet
   display: block;
   margin: ▶ 8px;
}
Inherited from html
html {                          user agent stylesheet
   color: -internal-root-color;
}
```

1.2 Ch01_inset01_02 Developer View

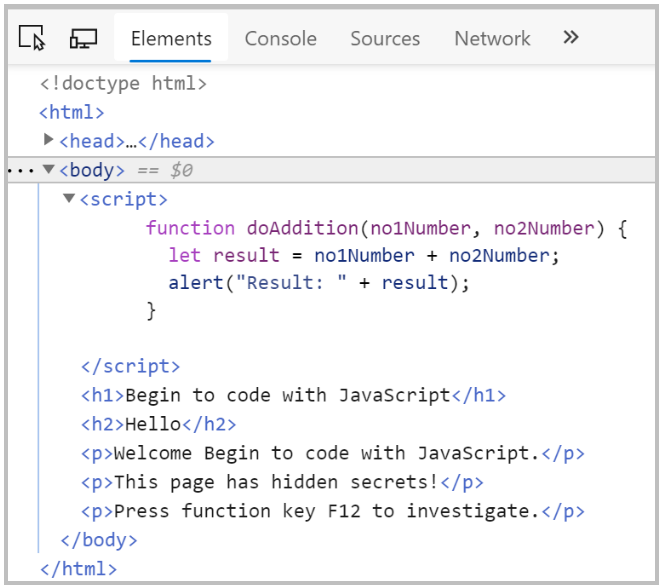
This is called *Developer View*. It shows all the elements that make up the web page. A complete description of everything you can do in this view would not fit in this book. Don't be worried about how complicated it all looks, we are only going to use a couple of the features. We are going to start by looking at the elements that make up the text on the page. Make sure that the Elements tab is selected as you can see in the figure above. Then look at the text underneath.



```
<!doctype html>
<html>
 ▶<head>…</head>
…▼<body> == $0
   ▶<script>…</script>
    <h1>Begin to code with JavaScript</h1>
    <h2>Hello</h2>
    <p>Welcome Begin to code with JavaScript.</p>
    <p>This page has hidden secrets!</p>
    <p>Press function key F12 to investigate.</p>
  </body>
</html>
```

1.3 Ch01_inset01_03 Page elements

The Figure above shows the elements on this page. You can see that the text that appears on the page is here. Parts of the text are enclosed in what look like formatting instructions, for example some is marked as <h1> and some as <p>. If you look back at the web page as displayed you will notice that the <h1> text is in a large heading font, whereas the <p> text is in smaller text. This is how pages are formatted. You can see how the page works, but where is the secret? To answer this, click the right-pointing arrowhead at the left of the word script to open this part of the view.



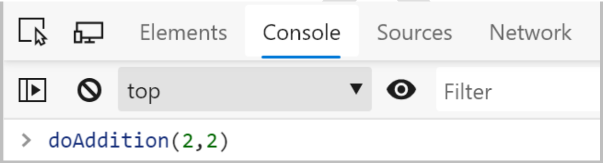
```
<!doctype html>
<html>
 ▶<head>…</head>
…▼<body> == $0
   ▼<script>
          function doAddition(no1Number, no2Number) {
            let result = no1Number + no2Number;
            alert("Result: " + result);
          }

    </script>
    <h1>Begin to code with JavaScript</h1>
    <h2>Hello</h2>
    <p>Welcome Begin to code with JavaScript.</p>
    <p>This page has hidden secrets!</p>
    <p>Press function key F12 to investigate.</p>
  </body>
</html>
```

1.4 Ch01_inset01_04 JavaScript function

Clicking the arrow opens that part of the listing. The hidden feature is a function called `doAddition`. This takes two numbers, adds them together and displays the result using an alert. Later in the text we will go into detail of how this JavaScript works, but even at this stage it is quite clear what is going on.

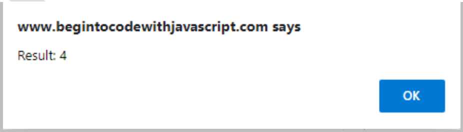
However, this function is never actually used in the webpage. We can use it ourselves by entering it into the JavaScript console which is built into the browser. This performs JavaScript statements that you type in. You can open the console by selecting the tab at the top of the window.



```
> doAddition(2,2)
```

1.5 Ch01_inset01_05 JavaScript console

This is the Console window. I've typed in the name of the function and given it two numbers to work on. When the function runs it display an alert with the result it has calculated.



www.begintocodewithjavascript.com says

Result: 4

1.6 Ch01_inset01_06 Result alert

We can use the JavaScript console to type in other JavaScript commands. You can use JavaScript to perform calculations by just entering them. When you press the Enter key the answer will be displayed. In fact, the console is often keen to give you an answer even before you press Enter. The console will also try to help as you type in by suggesting what you might be typing. You can accept any suggestion by using the cursor key to select the suggestion and then pressing the Tab key.

Code Analysis

We can learn a little about the way JavaScript works by giving the JavaScript console some commands and considering the responses.

```
> 2+3
```

This looks like a sum, and as you might expect, you get a number for an answer

```
<-5
```

We can repeat this with something other than a number:

```
> "Rob"+" Miles"
```

Some text enclosed in double quotes is interpreted by JavaScript as a string of text and it is perfectly happy to use + to add two strings together. Note that if you want to have a space between the two words in the sum you have to actually put it into the strings that you add together (in my example above there is a space in front of the 'M' in the second word.

```
<- "Rob Miles"
```

We can do other kinds of sums, for example we can subtract using minus (-).

```
>  6-5
```

This produces the result that you would expect.

```
<-  1
```

JavaScript seems quite happy when we ask it to do sensible things. Now, let's try asking it to do something stupid. What do you think would happen if we tried to subtract one string from another using the following statement?

```
> "Rob"-" Miles"
```

While it seems sensible to regard + as meaning "add these strings together" there doesn't seem to be a sensible interpretation of minus when you are working with strings. In other words, it is meaningless to try and subtract one string from another. If you enter this into the console you get a strange response from JavaScript

```
<-  NaN
```

The JavaScript console is not calling for grandma to come and sort the problem out. The value "NaN" means "not a number". It is a way that JavaScript indicates the result of a calculation has no meaning. Some programming languages would display an error message and stop a program if you tried to use them to subtract one string from another. JavaScript does not work like this. It just generates a result value that means “this result is not a number” and keeps going. We will consider how to a program can manage errors like this later in the text.

And since we are talking about errors, how about asking JavaScript to do some silly math:

```
>  1/0
```

When I got my first pocket calculator the first I tried to do with it was calculate one divided by zero. I was richly rewarded by a result that just kept counting upwards. What do you think JavaScript will do?

```
<-  Infinity
```

JavaScript says that the result of the calculation is the value “Infinity”. This is another special value that is generated by JavaScript when it does calculations. Talking of calculations, how about asking JavaScript to do another one for us.

```
>  2/10+1/10
```

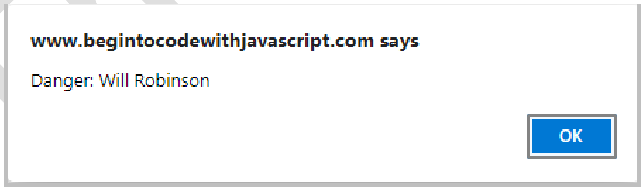
This calculation involves real numbers (i.e. ones with a fractional part). The calculation is adding 0.2 to 0.1 (a fifth to a tenth). This should produce the result 0.3 but what we get is interesting:

```
<-  0.30000000000000004
```

This number is very, very close to 0.3 (the correct answer) but is ever so slightly larger than it should be. This illustrates an important aspect of the way that computers work. Some values that we can express very easily on paper are not held exactly by the machine. This is only usually a problem if we start performing tests with the values that we calculate, for example a check to see if the calculated result above was equal to the value 0.3 might fail because of the tiny difference. Let’s see if we can use JavaScript to do some things for us. How about this:

```
>  alert("Danger: Will Robinson")
```

This statement doesn’t calculate a result. Instead it calls a function called alert. The function is provided with a string of text. It asks the browser to display the string as a message in an alert box.


www.begintocodewithjavascript.com says

Danger: Will Robinson

1.7 Ch01_inset01_07 Danger Will Robinson

This is how the `doAddtion` function displays the result it has calculated. Finally, lets try another function called print:

```
>  print()
```

What you will see next depends on the computer and the browser that you are using. But you should see a print window appear which offers you the chance to print the web page. If you've ever wondered what happens when you press the print button on a web page; you know now.

Congratulations. You now know how web pages work. Your browser fetches a file from the server and then follows the instructions in that file to build a page for you to look at. The file contains text that is to be displayed on the page along with formatting instructions. A page file can also contain JavaScript program code.

The instructions that the browser follows are expressed in a language called Hyper-Text Markup Language (HTML). In the next chapter we'll take a detailed look at HTML. But before we do that, we need to get some tools that will let us fetch the sample code for this book onto our computer and work with HTML and JavaScript.

Tools

You will need some software tools to get the best out of the exercises in this book. We are going to start with two, a program called "git" that will manage the program files that we work on and a program called "Visual Studio Code" which we will use to work on the files. Neither of these will cost you any money, and they are available for Windows, Macintosh and Linux based computers. You can follow through the printed instructions below, or you can use one of my Video Walkthroughs that you can find here:

https://www.begintocodewithjavascript.com/media

Programmer's Points

Git and Visual Studio Code are professional tools

When you learned to ride a bike you probably had one with training wheels. And people learning to drive a car usually start in something small and easy to handle. You are learning programming with the tools that professionals work with. This is a bit like learning to drive using a Formula 1 racing car. However, there is nothing to worry about here. A Formula 1 car might look a bit scary, but it still has a steering wheel and the usual set of pedals. You don't have to drive it fast if you don't want to and the consequences of a crash are much less.

GitHub and Visual Studio code have a huge range of features, but you don't have to use them. Just like there are buttons on my car dashboard that I don't press because I'm not sure what they do, you don't have to know about every feature of these tools to make good use of them.

It is very sensible to start developing with "proper" tools as recruiters are often as interested in the tools that you are familiar with as they are with the programming languages that you can work with.

Getting Git

The source code of programs that you write is stored on your computer as files of text. You work on your programs by changing the contents of these files. When I was starting out programming, I learned very quickly that you can go backwards as well as forwards when writing software. Sometimes I would spend a lot of time making changes to my programs that would turn out to be a bad idea and I would have to go back and undo them all. I solved this problem by making copies of my program code before I did any major edits. That way if anything went bad, I could go back to my original files.

Lots of other programmers noticed this problem too. They also noticed that if you release a program to users it is very useful to have a "snapshot" of that code so that you can keep track of any changes that you make. The best programmers are great at being "intelligently lazy" and so they created software to manage this. One of the most popular programs is called Git.

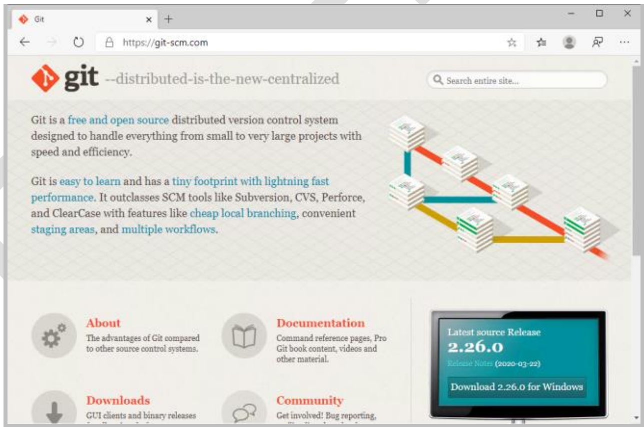
Git was created in 2005 by Linus Torvalds who was writing the Linux operating system at the time. He needed at tool that could track what he was doing and make it easy for him to work with other people. So he created his own. Git is a professional tool and very powerful. It lets large numbers of developers work together on a single project. Different teams can work on their own versions of the code which can them be merged. There is no need for you to use of all these powerful features though. You're just going to use Git to keep track of our work and as a way of obtaining the example programs.

Make Something Happen

Install Git

I'm going to give you instructions for Windows 10. The instructions for macOS are very similar. First you need to open your browser and visit the web page:

https://git-scm.com



git --distributed-is-the-new-centralized

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases

Community
Get involved! Bug reporting,

Latest source Release
2.26.0
Release Notes (2020-03-22)
Download 2.26.0 for Windows

2.1 Ch01_inset02_01 Git Install page

Follow the installation process selecting all the defaults.

Getting Visual Studio Code

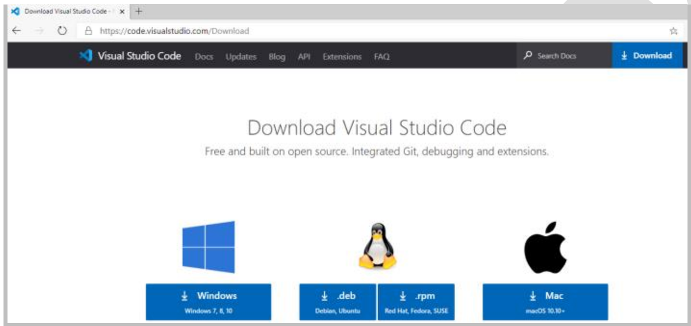
If you want to write a letter you would use a Word processor. To perform calculations, you might use a spreadsheet. Visual Studio Code is a tool that you can use to edit your program files. It can do a lot more than this, as we shall see later. But for now, we are going to use it as a super powerful program editor. Visual Studio Code is free.

Make Something Happen

Install Visual Studio Code

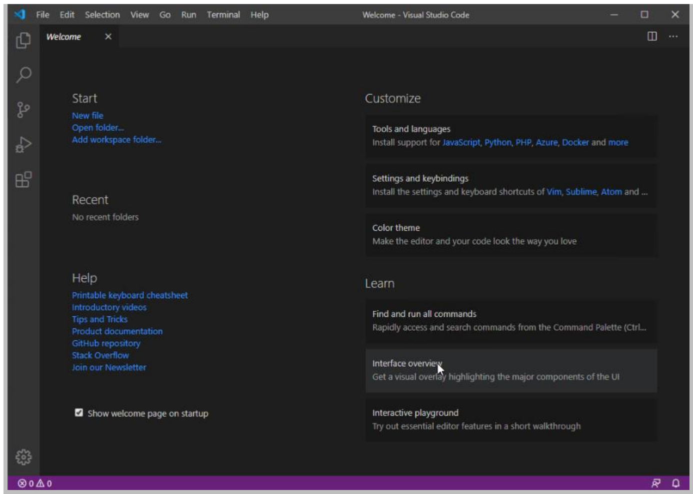
I'm going to give you instructions for Windows 10. The instructions for macOS are very similar. First you need to open your browser and visit the web page:

https://code.visualstudio.com/Download


Download Visual Studio Code
https://code.visualstudio.com/Download
Visual Studio Code
Docs
Updates
Blog
API
Extensions
FAQ
Search Docs
Download
Download Visual Studio Code
Free and built on open source. Integrated Git, debugging and extensions.
Windows
Windows 7, 8, 10
.deb
Debian, Ubuntu
.rpm
Red Hat, Fedora, SUSE
Mac
macOS 10.10+

3.1 Ch01_inset03_01 Visual Studio Code download page

Click the version of Visual Studio Code that you want and follow the instructions to install it. Once it is installed you will see the start page.


File Edit Selection View Go Run Terminal Help
Welcome - Visual Studio Code
Welcome
Start
New file
Open folder...
Add workspace folder...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, Python, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and ...
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl...
Interface overview
Get a visual overlay highlighting the major components of the UI
Interactive playground
Try out essential editor features in a short walkthrough

3.2 Ch01_inset03_02 Visual Studio Code start page

Now that you have Visual Studio installed the next thing you need to do is fetch the sample files to work on.

Getting the Sample Files

The sample programs, along with a lot of other stuff, are stored on *GitHub*. GitHub is a service that is underpinned by the Git system. You can store your own files on GitHub (and not just programs). You can also use GitHub to host web pages containing JavaScript programs that you create. This makes programs that you write accessible by anyone in the world. To do all this you will need to create a GitHub username and download some software onto your computer. We will create your username later. For now, we are going to just download the sample repository and edit hello.html, the file that we worked with at the start of this chapter.

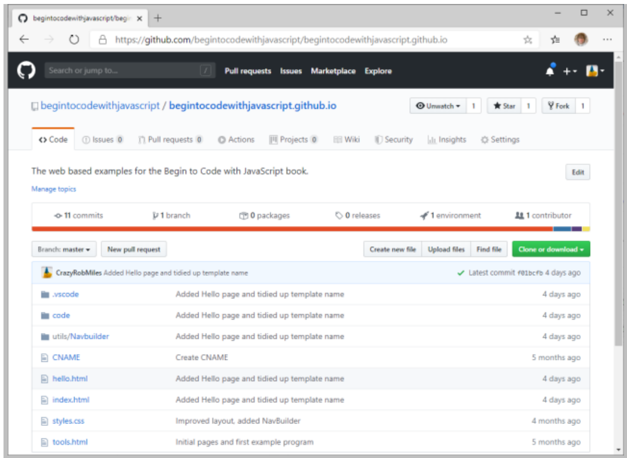
Make Something Happen

Clone the sample repository

A *repository* in Git is a collection of files. Whenever I start working on something new I create a repository to hold all the files I'm going to create. I've got a private repository that contains all the text of this book. And I've made a public repository to hold the sample files. Repositories on GitHub can be accessed directly from the browser. The sample files for this book are at the repository with this url (uniform resource locator):

https://github.com/begintocodewithjavascript/begintocodewithjavascript.github.io

If you visit this url with your browser you will find that you can navigate all the files, including the file "hello.html" that we investigated earlier and take a look at what is inside them.



The web based examples for the Begin to Code with JavaScript book.

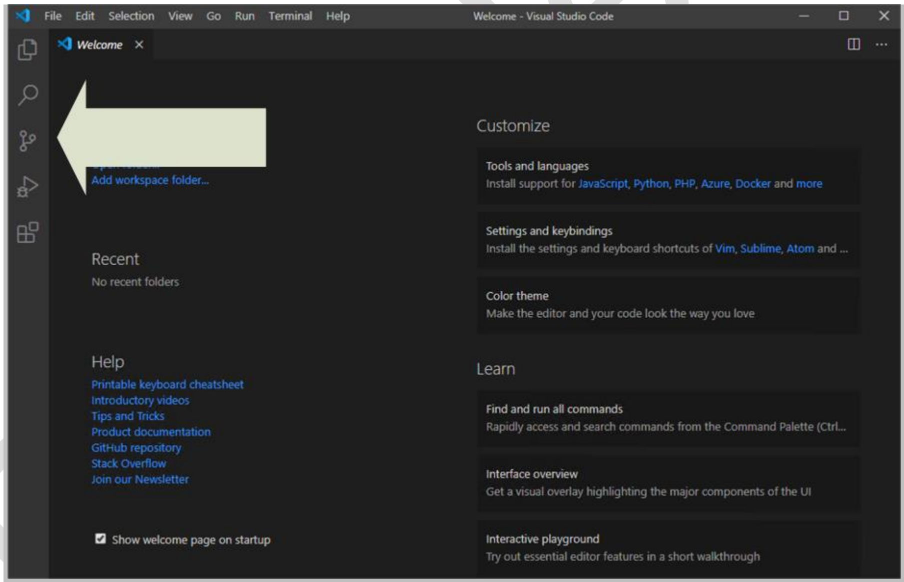
11 commits | 1 branch | 0 packages | 0 releases | 1 environment | 1 contributor

CrazyRobMiles Added Hello page and tidied up template name ✓ Latest commit f01bcf0 4 days ago

.vscode	Added Hello page and tidied up template name	4 days ago
code	Added Hello page and tidied up template name	4 days ago
utils/Navbuilder	Added Hello page and tidied up template name	4 days ago
CNAME	Create CNAME	5 months ago
hello.html	Added Hello page and tidied up template name	4 days ago
index.html	Added Hello page and tidied up template name	4 days ago
styles.css	Improved layout, added NavBuilder	4 months ago
tools.html	Initial pages and first example program	5 months ago

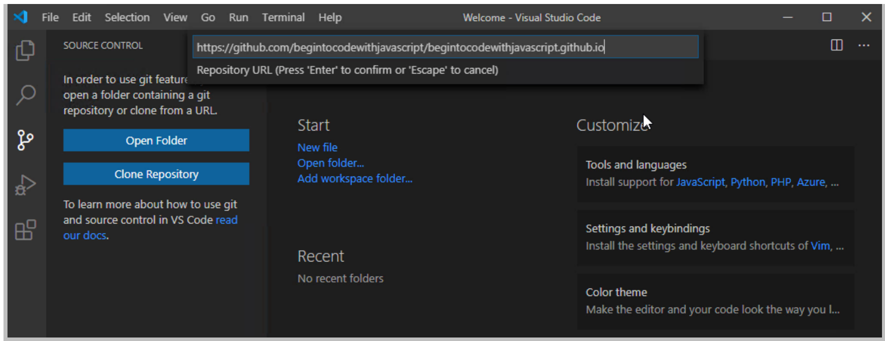
4.1 Ch01_inset04_01 Repository home page

You can see that GitHub is keeping track of the changes that I have made to the example programs. We are going to use Visual Studio Code to clone this repository.


File
Edit
Selection
View
Go
Run
Terminal
Help
Welcome - Visual Studio Code
Welcome
Add workspace folder...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, Python, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and ...
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl...
Interface overview
Get a visual overlay highlighting the major components of the UI
Interactive playground
Try out essential editor features in a short walkthrough

4.2 Ch01_inset04_02 Visual Studio Code Source Control Button

Start Visual Studio Code and click the Source Control button as shown on the figure above. This opens the Source Control dialog as shown below. Next click the Clone Repository button to begin the process of fetching a repository from GitHub.

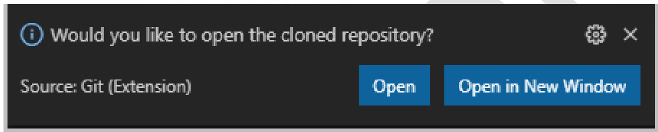

File Edit Selection View Go Run Terminal Help
Welcome - Visual Studio Code
SOURCE CONTROL
https://github.com/begintocodewithjavascript/begintocodewithjavascript.github.io
Repository URL (Press 'Enter' to confirm or 'Escape' to cancel)
In order to use git features, open a folder containing a git repository or clone from a URL.
Open Folder
Clone Repository
To learn more about how to use git and source control in VS Code read our docs.
Start
New file
Open folder...
Add workspace folder...
Recent
No recent folders
Customize
Tools and languages
Install support for JavaScript, Python, PHP, Azure, ...
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, ...
Color theme
Make the editor and your code look the way you l...

4.3 Ch01_inset04_03 Visual Studio Code Source Clone Repository

Visual Studio code is going to download the contents of the repository and store them on your machine. Enter the url of the repository in the dialog that appears. The url you want to use is:

https://github.com/begintocodewithjavascript/begintocodewithjavascript.github.io

When you press enter at the end of the url you will be asked where on your computer you want to put the files that are about to be copied. I suggest that you create a folder called GitHub in your “documents” folder and use that, but you can put the repository anywhere you like. Once you’ve selected the folder Visual Studio will copy all the files in the repository from the GitHub site onto your computer.



4.4 Ch01_inset04_04 Visual Studio Code Repository Clone Complete

When all the files have been copied Visual Studio Code will ask if you want to open the repository. Click Open to open it.

Congratulations, you have cloned you’re first repository! Later in the text you will discover how to create your own repositories to store your programs. Remember that you can use GitHub to store anything that you might want to work on, not just program files. If you have an assignment to write you could create a repository to hold the documents and images. This would be an even better idea if you were working on the assignment with other people as GitHub is a great collaboration tool.

Working on files with Visual Studio Code

We can round off this chapter by working the JavaScript program that we saw at the very start. The process we are going to follow will look like this:

1. Edit the program in the HTML file.
2. Save the file back to disk.

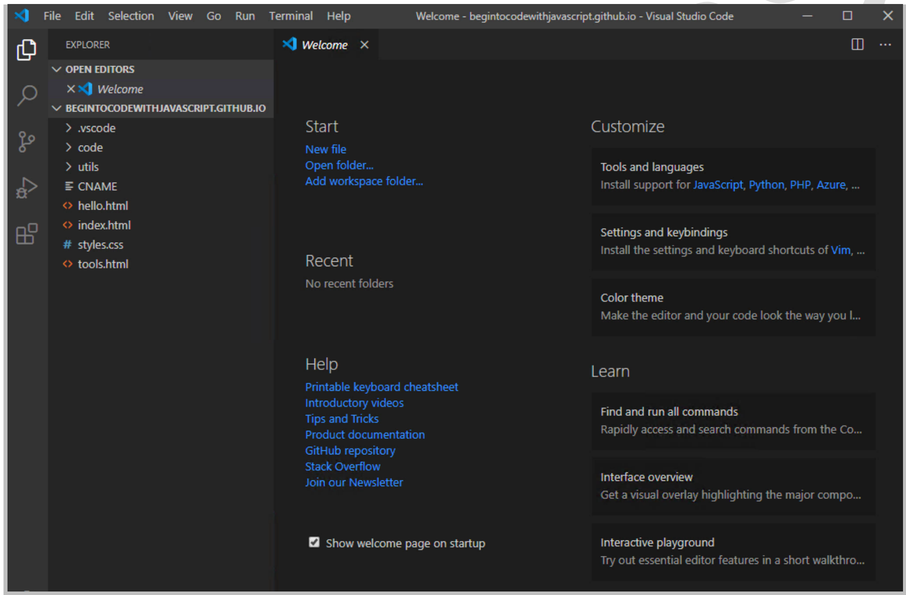
3. Use a web browser to view the HTML file and see what it does.

This is the process you will be using for a lot of the rest of the book. In the next chapter you will discover how to make your programs public so that anyone in the world can view them.

Make Something Happen

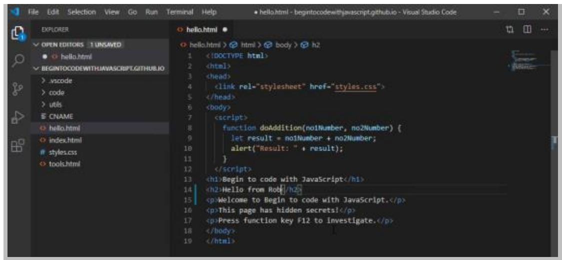
Edit the secret program

At the end of the last session you opened the example repository that you'd downloaded from GitHub. Now you get to edit the hello.html file that we saw contains the secret program.


File Edit Selection View Go Run Terminal Help
Welcome - begintocodewithjavascript.github.io - Visual Studio Code
EXPLORER
OPEN EDITORS
Welcome
BEGINTOCODEWITHJAVASCRIPT.GITHUB.IO
.vscode
code
utils
CNAME
hello.html
index.html
styles.css
tools.html
Welcome
Start
New file
Open folder...
Add workspace folder...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, Python, PHP, Azure, ...
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, ...
Color theme
Make the editor and your code look the way you l...
Learn
Find and run all commands
Rapidly access and search commands from the Co...
Interface overview
Get a visual overlay highlighting the major compo...
Interactive playground
Try out essential editor features in a short walkthro...

5.1 Ch01_inset05_01 Visual Studio Code Example Repository

The Explorer window at the left hand side of the Visual Studio Code window provides a view of all the files and folders in the repository. You can click the ">" in front of folders in the Explorer view to open them and view their contents. For now, you are just going to look in the hello.html file, so click the filename hello.html in the Explorer to open it.

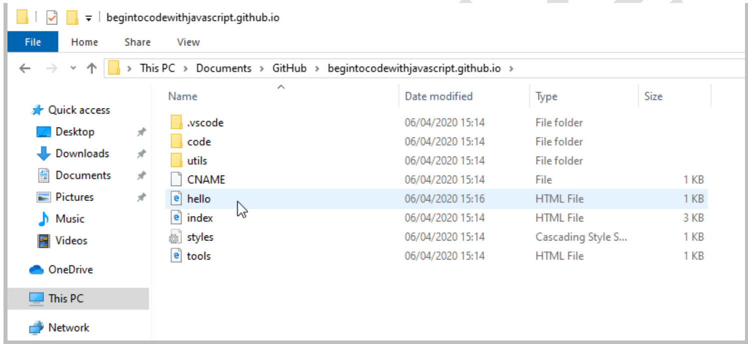


```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <script>
    function doAddition(no1Number, no2Number) {
      let result = no1Number + no2Number;
      alert("Result: " + result);
    }
  </script>
<h1>Begin to code with JavaScript</h1>
<h2>Hello from Rob</h2>
<p>Welcome to Begin to code with JavaScript.</p>
<p>This page has hidden secrets!</p>
<p>Press function key F12 to investigate.</p>
</body>
</html>
```

5.2 Ch01_inset05_02 Visual Studio Code Editing hello.html

Once the page has opened you can make some changes to the text in the file. I've changed one heading so that it says "Hello from Rob". You can save the file by holding down the control key and pressing S (CTRL+S). Or you could use the Save command. Either way, you now want to view the changed file in a browser to see if the changes have worked.

When you cloned the repository, you told Visual Studio Code where to put the files, so now is the time to open File Explorer and navigate to that folder. If you've forgotten where you put the files you can find out just by resting your mouse pointer over the filename in Explorer. Visual Studio Code will then show you the path to that file.

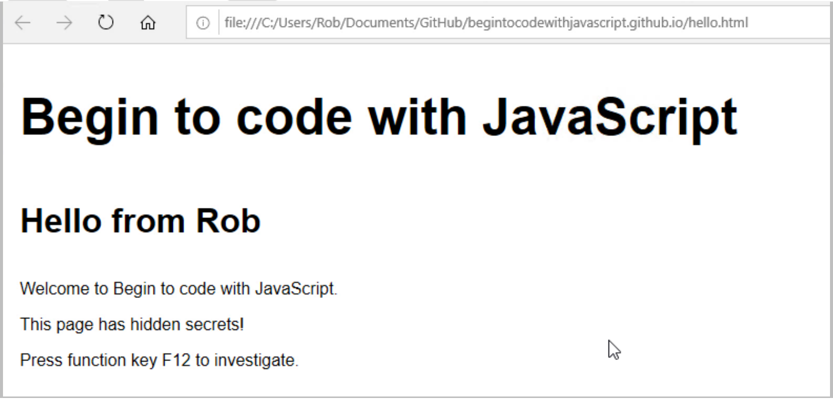


This PC > Documents > GitHub > begintocodewithjavascript.github.io

Name	Date modified	Type	Size
.vscode	06/04/2020 15:14	File folder	
code	06/04/2020 15:14	File folder	
utils	06/04/2020 15:14	File folder	
CNAME	06/04/2020 15:14	File	1 KB
hello	06/04/2020 15:16	HTML File	1 KB
index	06/04/2020 15:14	HTML File	3 KB
styles	06/04/2020 15:14	Cascading Style S...	1 KB
tools	06/04/2020 15:14	HTML File	1 KB

5.3 Ch01_inset05_03 Finding the hello.html file

If we double-click this file it will be opened by the browser.



file:///C:/Users/Rob/Documents/GitHub/begintocodewithjavascript.github.io/hello.html

Begin to code with JavaScript

Hello from Rob

Welcome to Begin to code with JavaScript.

This page has hidden secrets!

Press function key F12 to investigate.

Ch01_inset05_04 Browsing the hello.html file

Now you will see the file in all its edited glory. Note that the address being browsed is now a file on your local storage, rather than on the web. Note also that you can press F12 if you like and view the contents of the file just like we did at the start of this chapter.

What you have learned

You might feel that you've spent a lot of this chapter just following instructions, but actually you've learned rather a lot. You've discovered that JavaScript is a programming language, providing a means by which you can tell a computer how to do something. You've had a conversation with JavaScript itself. You've learned that looking after the source files of your programs is important, although great programmers sometimes think of very silly names (for example "git") for their programs sometimes. You've installed the git system and your program editor, Visual Studio Code. Finally, you've copied all the example code onto your machine by "cloning" the repository held on GitHub and even managed to edit one file and view the effects in your browser.

To reinforce your understanding of this chapter, you might want to consider the following "profound questions" about JavaScript, computers, programs, and programming.

What does the word "script" mean in the name JavaScript?

The word "script" in the name refers to the way that JavaScript programs were intended to run. The browser would read each JavaScript statement and then perform it; just like an actor would act out the script of a play. This is not how all programming languages work. Some programming languages are designed to be *compiled*. This means that the source code of the program is converted into the low-level instructions that are run by the computer hardware. These low-level instructions are then directly obeyed by the hardware to make the program run.

Compiled languages run faster than scripts because when the compiled program runs the computer doesn't have to put any effort into working out what the program source is doing, it can just obey the low-level instructions. However, you need to make a different version of the compiled code for each different type of computer. For example, a compiled file for a Windows PC would not run on a Raspberry Pi.

JavaScript was intended to perform simple tasks inside a browser, so it was created as a scripting language. However, it has now become so popular that modern browsers compile JavaScript before running it so that it runs as quickly as possible.

Does my JavaScript programs run on the Web Server?

No. The job of a web server is just to serve up files. The browser (the program running on the user's computer) is responsible for actually creating the display of a web page and running any JavaScript programs in that page.

Do JavaScript programs run at the same speed on all computers?

No. The faster the host computer, the faster the browser (and the JavaScript programs it is hosting) will run.

Do JavaScript programs run faster if I have a faster network connection?

No. A faster network connection will improve the speed at which the JavaScript programs will be loaded into the browser, but the actual speed the JavaScript program runs is determined by the speed of the host computer. Having said that, if the JavaScript program uses the host computer network connection these actions will of course happen more quickly.

Can we view the JavaScript programs in every page we visit?

Yes, you can. The F12 trick (pressing F12 when viewing a web page in a browser) will opens the development view of the page. You can use this to view the JavaScript source code in the page. If you are concerned about someone copying the JavaScript code you can use a tool called an *obfuscator* which is a piece of software that changes the appearance (but not the behavior) of a program so that it is very hard to understand. Take a look at https://www.javascriptobfuscator.com/ for more details.

How big can a JavaScript program be?

A JavaScript program can be very large indeed. Modern web browsers are very good at handling large programs and the speed of modern networks means that the code can be downloaded very quickly. Some people have even created complete computer emulations in JavaScript that you can run in a browser.

Can you run JavaScript outside a web browser?

Yes you can. Some web pages can be converted into applications which then run on the local computer. There are also ways in which a computer can be made to host JavaScript applications in the same way that a browser does. We will look at these later in the text.

Why is "Git" called "Git"?

This is probably the hardest question in this book. In the UK the word "git" is a form of mild abuse. You would call someone a git if they spilled your drink on purpose. It seems that Linus Torvalds called his first version of the program "His stupid content tracker" and then hit upon the word git as a shorter version of this.

Can I do private work on GitHub?

Yes. GitHub is very popular with programmers who are working on Open Source projects, but you can also make a GitHub repository private so that only you can see it. If you use the free subscription the number of private repositories you can create is limited, as is the number of people you can work with on a shared project.

What do I do if I "break" my program?

Some people worry that things they do with a program on the computer might "break" it in some way. I used to worry about this too, but I've conquered this fear by making sure that whenever I do something, I always have a way back. Git and GitHub are very useful in this respect. Later in the text we will discover how to use GitHub to take "snapshots" of projects which we can return to if we break our program. We can also use the Git desktop program to search for changes that we have made to the files in a project.

Why is the Visual Studio Code display of the hello.html file in different colors?

This is called *source code highlighting.* Visual Studio Code has a list of words that are “special” as far as JavaScript and HTML are concerned. These special words are called *keywords*. For each keyword Visual Studio has a characteristic color, in the case of Visual Studio Code keywords are displayed in blue, functions are displayed in yellow, strings of text are orange and everything else is white. The intention is to make it easier for programmers to understand the structure of the program. Note that there is nothing in the program file that specifies the color of each element, this is something that Visual Studio Code does.

Will “artificial intelligence” mean that one day we won’t have to write programs?

This is a very deep question. To me, artificial intelligence is a field where lots of people are working very hard to make a computer really good at guessing. It turns out that by giving computers lots of information, and telling them how the information is related, a program can then use all this stuff to make a pretty good guess as to the context of a statement.

I suppose that all humans do is “guess” at the meaning of things. Maybe one day a doctor really will want me to drink a hot bath before I take my medicine (see the instructions above), in which case I’ll do the wrong thing. However, humans have a much greater capacity to store experiences and link them together, which puts the computer at a distinct disadvantage when it comes to showing intelligence. Maybe in time this will change. We are already seeing that in specific fields of expertise, for example finance and medical diagnosis, artificial intelligence can do very well.

However, in my opinion, when it comes to telling the computer exactly what we want them to do, we’ll be needing programmers for quite a long time. Certainly, long enough for you to pay off your mortgage.

2

HyperText Markup Language (HTML)

What you will learn

In the previous chapter you learned that a JavaScript program can live inside a web page. You saw that the file hello.html had a secret script inside it. In this chapter we will find out more about the HTML standard that tells the browser program what a web page should look like. Then we'll discover how to link JavaScript to elements on a web page to allow our programs to interact with the user.

HTML and the World Wide Web

The first version of a *HyperText Markup Language* (HTML) was created in 1989 by Tim Berners-Lee. He wanted to make it easier for researchers to share information. At the time research reports were written as individual documents. If a document you were reading contained a *reference* to another you would have to go and find the other one. Tim Berners-Lee designed a system of computer *servers* that share electronic copies of documents. A document could contain *hyper links* to other documents. Readers used a *browser* program to read the document from the servers and follow the links from one document to another. These documents are called *HyperText* documents and the language that described their contents is called the *HyperText Markup Language* or HTML.

Tim Berners-Lee also designed a protocol to manage the transfer of HTML formatted documents from the server into the browser. This standard is called the *Hyper Text Transfer Protocol* or "HTTP". There is now also secure version of this protocol called "HTTPS" which add security to the web. HTTPS allows a browser to confirm the identity of a server and it also protects messages sent between the server and the browser to prevent eavesdropping. The HTTPS protocol is what makes it possible for us to use the world wide web for banking and e-commerce.

In 1990 the first system was released as the "World Wide Web". The documents that you could download were called "web pages". The sever hosting the web pages was called a "web site". In 1993 Marc Andreeson added the ability to display images in web pages and the web became extremely popular.

The World Wide Web was designed to be extensible and for many years different browser manufacturers added their own enhancements to the standards, leading to problems with compatibility where web sites would only work with specific browser programs. Recently the situation has stabilized. The World Wide Web Consortium (W3C) now sets out standards which are implemented by all browser manufacturers. The latest standard, HTML 5, is now very stable and is the version used in this book.

Fetching web pages

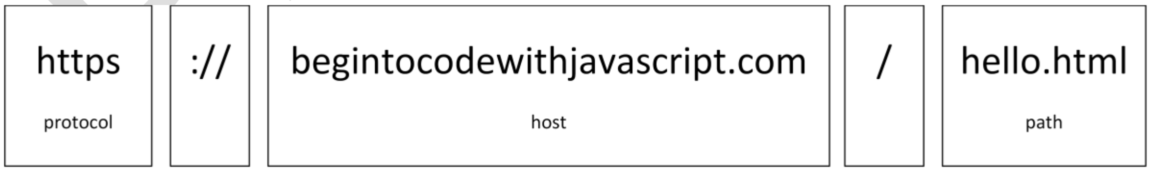
The location of the page on a server is given using a *uniform resource locator* or url. This has three elements:

- The *protocol* to be used to talk to the site. This sets out how a browser asks for a web page, and how the server replies. Web pages use HTTP and HTTPS. HTTP stands for "HyperText Transport Protocol" and HTTPS is the secure version of this protocol.

- The *host*. This gives the address of the server on the network. The world wide web sits on top of a networking protocol called TCP/IP (Transport Control Protocol/Internet Protocol) and this is the address on that network of the system that holds the web site you want to connect to.

- The *path*. This is the path on the host to the item that the browser wants to read.

You can see all these elements in the url that we used to access the hello page in Chapter 1.



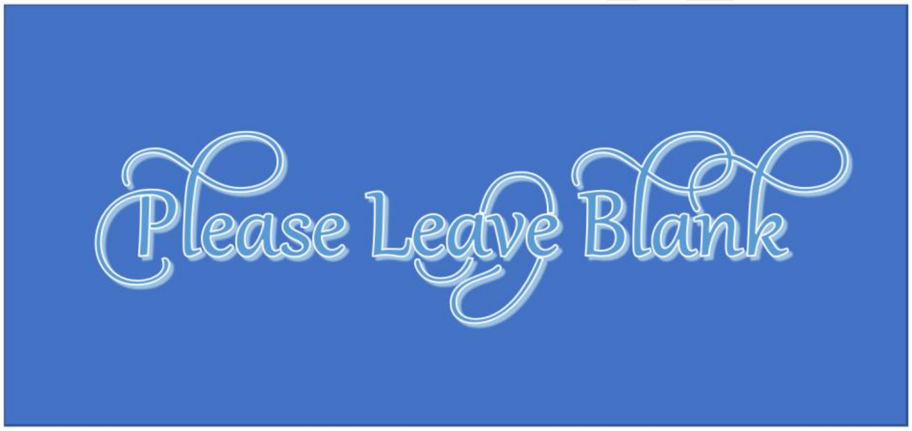
1. Figure 2.1 Ch02_Fig_01 URL structure

This url specifies that the site uses the secure version of the hypertext transfer protocol, that the address of the host server is "begintocodewithjavascript.com" and that the path to the file containing the web page is "hello.html". If you leave off the path the browser will automatically request a file with the path “index.html”. Most browsers will now automatically fill out the “https://” when you type in a web address. If the path is omitted from the url the server will send the contents of a file called “index.html”, which is called the *index page* of a site.

When the user requests a web site the browser sends a message to the server to request the page. This message is formatted according to the HyperText Transport Protocol (HTTP) and is often called a “get” request (because it starts with the word “GET”). The server then sends a response which includes status code and then, if it is available, the text of the web page itself. If the page cannot be found on the server (perhaps because the url was not given correctly) the HTTP status code results in the familiar “404 page not found” message. We will learn more about this process later in the book when we write some JavaScript code that gets web pages.

What is HTML?

This is not a guide to HTML. You can buy whole books that do very thorough job of describing the language and how it is used. But you should finish this section with a good understanding of the fundamentals. HTML is a *markup* language. That is what the “M” in HTML stands for. The word “markup” comes from the printing profession. Printers would be given text that had been “marked up” with instructions such as “print this part in large font” and “print this part in italic”.



2. Figure 2.2 Ch02_fig02 Please Leave Blank

Figure 2.2 above shows what happens if you don’t use a markup language properly. The customer wanted a cake with no writing on it. They said “Please Leave Blank” when asked what they wanted written on the cake. Unfortunately, the baker took this instruction literally. This kind of miss-understanding is impossible with HTML. The language has a rigid separation between the text that is to be displayed and the formatting instructions. In HTML, if I want something to be *emphasized,* I will use an HTML markup command to request this:

```
<em>This text is emphasized.</em> This text is not.
```

The sequence <em> is recognized by the browser as meaning “make the text that follows this instruction look slightly different from the other text”. It is called a *tag.* The browser will display emphasized text until it sees the sequence </em> which marks the end of the emphasized text. Most browsers emphasize text by displaying it as *italic*. If we viewed the above HTML in a Microsoft Edge we would see something that looks like this:

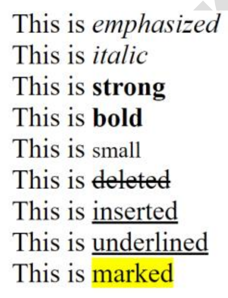
```
This text is emphasized. This text is not.
```

Once you understand the fundamentals of HTML it you can use it to format text. The HTML below shows a few more tags.

```
This is <em>emphasized</em><br>
This is <i>italic</i><br>
This is <strong>strong</strong><br>
This is <b>bold</b><br>
This is <small>small</small><br>
This is <del>deleted</del><br>
This is <ins>inserted</ins><br>
This is <u>underlined</u><br>
This is <mark>marked</mark><br>
```

Ch02-01 Text format tags

The example HTML above uses a tag, <br>, which means “take a new line”. The <br> tag does not need to be matched by a </br> element to “close it off”. This is because it has an immediate effect on the layout, it is not “applied” to any specific items on the page. When I pass this text into a browser, I get the following output.



This is *emphasized*
This is *italic*
This is **strong**
This is **bold**
This is small
This is ~~deleted~~
This is inserted
This is underlined
This is marked

3. Ch02_fig03 HTML text modification

If you look closely at the text in Figure 2 you will notice that some of the requests have similar results. For example, the emphasized and italic formats both produced italic output. The bold, italic and underline tags are regarded as slightly less useful than the more general ones such as “emphasized” or “strong”. The reasoning behind this is that if a display has no way of producing italic characters a request to display something in italic is not going to work.

However, if the display is asked to “emphasize” something it may be able to do this in a different way, perhaps by changing the color of the text. Output produced by HTML is intended to be displayed in a useful way on a huge range of output devices. When you use a markup language you should be thinking about the effect you want to add to a piece of text. You should think “I need to make this stand out; I’ll use the ‘strong’ format” rather than just making the text bold.

You can write the commands using upper or lower case, or any combination. In other words the tags `<em>`, `<EM>` and `<Em>` are all regarded as the same thing by the browser.

Display symbols

By now you should have a good idea how HTML works. A tag `<blah>` marks the start of something. The sequence `</blah>` marks the end. The tags can be nested, (i.e. placed inside each other).

```
<em>This is emphasized <strong>This is strong and emphasized</strong></em>
```

This HTML would generate:

```
This is emphsized This is strong and emphasized
```

For every start tag `(<blah>)` that marks a formatted area of text there should be a matching end tag `(</blah>)`. Most browsers are quite tolerant if you get this wrong, but the display that you get might not be what you want.

The question you are probably asking now is “How I can ever get to display the < (less than) and > (greater than) symbols in my web pages?” The answer is that HTML uses another character to mark the start of a *symbol entity*. The & character marks the start of a symbol. Symbols can be identified by their name:

```
This is a less than: &lt; symbol and this is a greater than &gt; symbol
```

The name of the less than character (<) is “lt” and that of greater than (>) is “gt”. Note that the end of a symbol name is marked by a semi-colon (;). If you are now wondering how we display an ampersand (&) the answer is that it has the symbol name amp.

```
This is an ampersand: &amp;
```

You can find a handy list of symbols and their names here: https://dev.w3.org/html5/html-author/charref Note that when you give a symbol name the case of the names is significant.

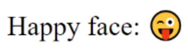
```
&Eacute;<br>
&eacute;<br>
```

The HTML above would display the upper (É) and lower (é) case versions of “e acute”. If you like emojis (and who doesn’t) you can add these to your web pages by using a symbol that includes the number of the emoji that you want.

```
Happy face: &#128540;<br>
```

Ch02-02 HTML Symbols

This will display a happy face.



Happy face: 😜

4. Ch02_fig04 Happy face

If you want to discover all the numbers that you can use to put emojis in your web pages, take a look here: https://emojiguide.org/

Lay out text in paragraphs

We now know how to format text. Next we must consider how we can lay this text out on the page. When HTML text is displayed the original layout of the text input is ignored. In other word, consider the text.

```
Hello
 world

    from   Rob
```

The layout of this text is a bit of a mess. However, when this text is displayed by a browser you see the following:

```
Hello world from Rob
```

The browser takes in the original text, splits it into words and then displays the words with single spaces between them. Any layout information in the source text is discarded. This is a good idea because the designer of a web page can't make any assumptions about the display that will be used. The same page needs to work on large and small displays, from smartphones to large LCD panels.

We've seen that the `<br>` sequence asks the browser to take a new line during the display of text. Now we are going to consider some more commands that control how text is laid out when it is displayed. The `<p>` and `</p>` commands enclose text that should appear in a paragraph.

```
<p>This is the first paragraph</p>
<p>This is the second paragraph</p>
```

This HTML will display two paragraphs.

```
This is the first paragraph
This is the second paragraph
```

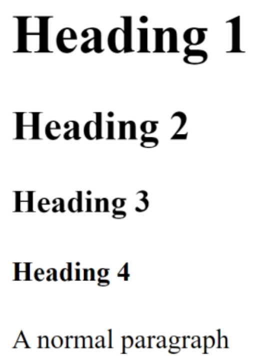
The `<br>` command is not the same as the `<p>` command; it does not space the lines out like a paragraph would.

Create headings

We can use other tags to mark up text as headings at different levels:

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<p>A normal paragraph</p>
```

We can use these in documents to create headings.



Heading 1

Heading 2

Heading 3

Heading 4

A normal paragraph

5. Ch02_fig05 Headings

You can use headings to create structure in a document.

Use pre-formatted text

But sometimes you might have something that you have already formatted. In this case you can use the `<pre>` tag to tell the browser not to perform any layout:

```
<pre>
This text
  is rendered
    exactly how I wrote it.
</pre>
```

The text enclosed by the `<pre>` tags is displayed by the browser without any changes to the formatting.

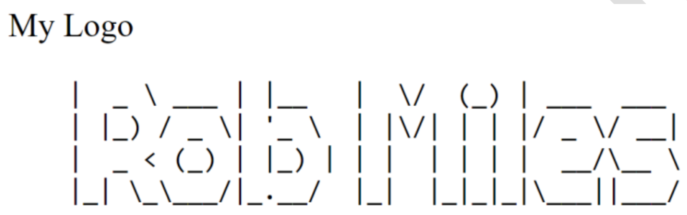
```
This text
  is rendered
    exactly how I wrote it.
```

The browser uses a *monospaced* font when displaying pre-formatted text. In a monospaced font all the characters have the same width. Many fonts, including the one used to print this paragraph, are *proportional*. This means that each character has a particular width, for example the "I" character is much smaller than the "m" character. However, for some text, for example ASCII art, it is important that all the characters line up. This logo would not look correct if it was not displayed with a monospaced font.

```
<p> My Logo</p>
<pre>
|  _ \ ___ | |__   |  \/  (_) | ___  ___
| |_) / _ \| '_ \  | |\/| | | |/ _ \/ __|
|  _ &lt; (_) | |_) | | |  | | | |  __/\__ \
|_| \_\___/|_.__/  |_|  |_|_|_|\___||___/
</pre>
```

Ch02-03 A pre-formatted logo

Note that the ASCII art above contains a "<" character. I've had to convert this to a symbol (&lt;) so that it is displayed correctly. This is important. Remember that the browser will not format pre-formatted text, but it still observes the character conventions that you must use to display characters and symbols. You can add tags to the pre-formatted text to make parts of it emphasized. You can even put <p> tags inside preformatted blocks of and they might work, but this is not advised because it makes your html *badly formed*.



My Logo

```
|  _ \ ___ | |__   |  \/  (_) | ___  ___
| |_) / _ \| '_ \  | |\/| | | |/ _ \/ __|
|  _ < (_) | |_) | | |  | | | |  __/\__ \
|_| \_\___/|_.__/  |_|  |_|_|_|\___||___/
```

6. Ch02_fig06 My logo

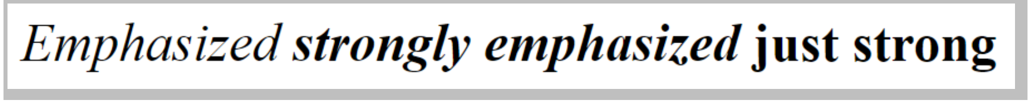
Programmer's Point

Don't abuse the browser

Browsers are generally very tolerant of badly formatted HTML. The browser will try to display something even if the HTML it receives is badly formatted. This means you can get away with HTML like this:

```
<em>Emphasized <strong>strongly emphazised</em> just strong </strong>
```

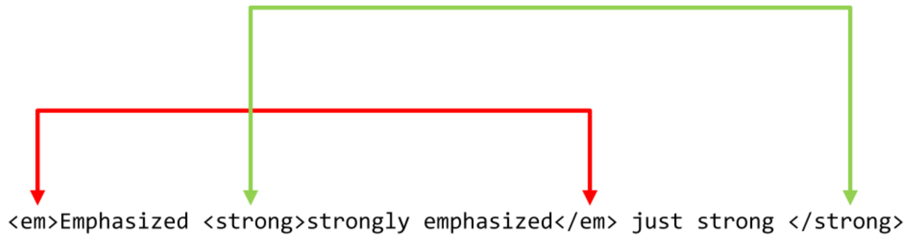
The browser will display what you would expect:



Emphasized ***strongly emphasized*** **just strong**

Ch02_Readeraid_01_Fig_01 emphasized text

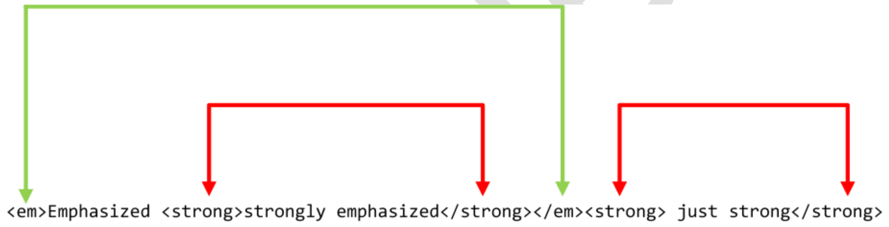
However, is *malformed HTML*. You might be wondering why. Let's take a look at the sequence of tags in the text.



<em>Emphasized <strong>strongly emphasized</em> just strong </strong>

Ch02_Readeraid_01_Fig_01 Bad Nesting

This is an example of what is called *bad nesting.* This is because the <em> tag "ends" inside the <strong> tag. In properly formed HTML a tag that is created inside another will end before the enclosing tag ends. The complete sequence of a start tag, text and end tag is called an *element.* While one element can completely contain another, it is not correct for elements to *overlap* like the ones in the above figure. The correct version of this HTML is shown below. Note that each element is complete.



<em>Emphasized <strong>strongly emphasized</strong></em><strong> just strong</strong>

Ch02_Readeraid_01_Fig_02 Good Nesting

The above figure shows what correct nesting looks like. Each element ends before its enclosing one. The reason why I'm stressing this is that most browsers can work out the meaning of the badly nested HTML and display it correctly, but some might not. This could lead to your web pages looking wrong to some people. I'm sure you've had the experience of having to switch browser because a particular web site doesn't look right. Now you know how this can happen.

You can identify incorrectly nested HTML when you see an end tag that doesn't match the most recent start tag. If you want to use a program to make sure that your HTML is correct you can use the official validator site here https://validator.w3.org/. You can point the validator at a site you have created or paste HTML text into it for checking.

Add comments to documents

You can add comments to an HTML document by enclosing the comment text in the sequences `<!--` and `-->` as follows:

```
<!-- Document Version 1.0 created by Rob Miles -->
```

The author credit would not be displayed by the browser, but you could view it in the source code by pressing the F12 key to open the developer view. As we go through this text I'll be telling you regularly how useful it is to add comments to your work, so I think it is a good idea to start doing this now.

Add images to web pages

For the first few years of its life the World Wide Web didn't have any pictures at all. The image tag was added by Marc Andreeson, one of the authors of Mosaic, the most popular browser in the early days of the web. The image tag contains the name of a file that contains an image:

```
<img src="seaside.JPG">
```

The image tag uses an *attribute* to specify the file that contains the image to be displayed. An attribute is given inside the tag as a name and value pair, separated by the equals character. When the browser finds an img tag it looks for the src attribute and then looks for an image file with that name. In the case of the above HTML the browser would look for an image called "seaside.JPG". It would look in the sample place on the server from which it loaded the web page. We must make sure that file exists on the server, otherwise the image will not be displayed.

What Could Go Wrong? – Beware of faulty filenames

The src attribute in an img tag is followed by the name of the file that is to be fetched from the server. While HTML doesn't care about the capitalization of tags (you can write IMG, img or Img for the tag name) the computer fetching the image file might. Some computers will deliver a file stored as "seaside.jpg" if you ask for one called "seaside.JPG". Others will complain that the file is not available.

I normally encounter this problem when I take a web site off my PC (where it has been working perfectly) and place it on the server (when all the image files suddenly vanish).

You can add another attribute to an img tag that gives alternative text that is displayed if the image cannot be found.

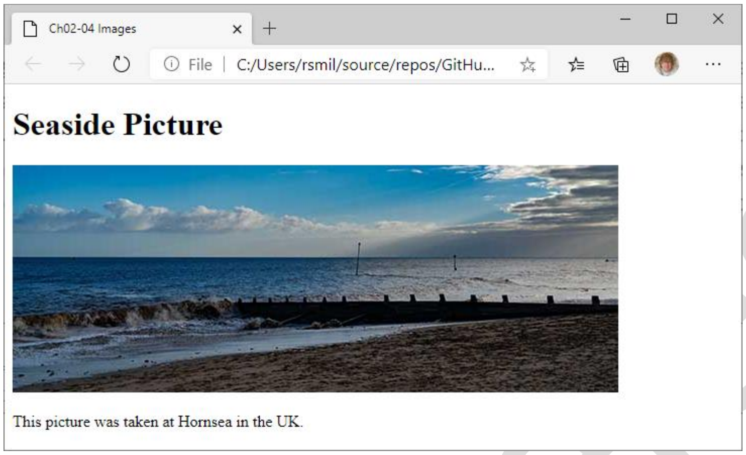
```
<img src="seaside1.JPG" alt="Maybe Rob got the filename wrong">
```

Now the browser will display the text "Maybe Rob got the filename wrong" if the image can't be located.

The image will be displayed in line with the text on the page. We can use the HTML layout tags to lay an image out sensibly with the surrounding text.

```
<h1>Seaside Picture</h1>
<p><img src="seaside.JPG"></p>
<p>This picture was taken at Hornsea in the UK.</p>
```

Ch02-04 Images



Seaside Picture

This picture was taken at Hornsea in the UK.

7. Ch02_fig07 Image

This image is 600 pixels wide. A pixel (short for *picture cell)* is one of the dots that make up the picture. The more pixels that you have the better looking the picture is. However, this can cause problems if the picture is too large to fit on the device being used to display the image. The `image` tag supports `width` and `height` attributes that can be used to set the displayed size of an image. So, if I want to display the image as 400 pixels wide I can do this:

```
<p><img src="seaside.JPG" width=400>
```

Note that I didn't specify the height, in which case the browser will automatically calculate the height that matches a width of 400 pixels. You can specify both height and width if you like, but you need to be careful not to make the pictures distorted. Setting the absolute width of an image using height and width attributes looks like a good idea at first but it can be restricting. Remember that an underlying principle of the World Wide Web is that a page should display in a useful way on any device. An image size of 400 pixels might be fine for a small device, but it will appear very small if viewed on a large TV display. In the next chapter we will discover how we can use stylesheets to allow a items on a webpage to be automatically scaled for the target device.

The HTML document

We now know that we can use tags to mark regions of text as needing to be formatted in a particular way; for example, `<em>` for emphasized text. We can also mark regions of text as being in paragraphs or levels of headings. We can apply several tags to a given piece of text to allow formatting instructions to be layered on top of each other, but we need to make sure that these instructions are properly "nested" inside each other. Now we can consider how to create a properly formatted HTML document. This is comprised of several sections:

```
<!DOCTYPE HTML>[1]
<html lang="en">[2]
  <head>[3]
       <!-- Heading here --!>
  </head>[4]
  <body>[5]
       <!-- Body text here --!>
  </body>[6]
</html>[7]
```

The browser looks for the sequence `<!DOCTYPE HTML>` at the start to make sure that it is reading an HTML file. All the HTML that describes the page is given between `<html>` and `</html>` tags. The `</html>` tag contains a `lang` attribute that specifies the language of the page. The language `"en"` is English. The `<head>` and `</head>` tags mark the start and end of the *heading* of the document. The heading contains information about the content of the page including styling information (of which more next chapter). The text in-between the `<body>` and `</body>` tags is what is to be displayed. In other words; everything we have learned up to now goes into the body part of the web page file.

[1] Indicates that this is an HTML document

[2] HTML tab with a language attribute

[3] Start of the heading of the web page

[4] End of the heading

[5] Start of the body text of the web page

[6] End of the body text

[7] End of the HTML text

Linking HTML Documents together

An HTML document can contain elements that link to another document. The other document can be on the same server or it can be on a different server entirely. A link is created by using an "a" tag which has an `href` attribute that contains the url of the destination page.

```
Click on <a href="otherpage.html">this link</a>to open another page.
```

The text in the body of the <a> tag is the text that the browser will highlight as the link. In the example HTML above the words "this link" will be the linkable text. This will result in text on the page that looks like this:

```
Click on this link to open another page.
```

If the reader clicks the link the browser will open a local file, in this case called `"otherpage.html"`, which will be displayed. The destination of the link can refer to a page on a completely different site:

```
<p>Click on <a href="https://www.robmiles.com"> this link</a> to go to my blog.</p>
```

Ch02-05 References

Making active Web Pages

There are lots of other things that I could tell you about HTML. The language can be used to create numbered and un-numbered lists and tables. However, this is not a book about HTML, it is about programming. What we want is a way of getting JavaScript code to run inside our web page. Then we can start exploring the language.

You already know that a JavaScript program can sit alongside an HTML page design. You saw that in Chapter 1 when you used the Developer View (obtained with F12) in the browser to take look at the hidden program inside the web page hello.htm. That HTML file contained a `script` element holding some JavaScript code. We used the console to run a JavaScript function. Now we are going to trigger a function by pressing a button.

Using a button

```
<button onclick="doSayHello()">Say Hello</button>
```

One way to create an active web page is by using a button. The HTML above creates a button that contains the text “Say Hello”. The button is displayed in the normal flow of the text in the page.

8. Ch02_fig08 Say Hello button

The button has an `onclick` attribute. One of the great things about JavaScript is that most of the time the names make sense. The `onclick` attribute specifies a function that is to be used when the button is clicked. In this case the attribute specifies a JavaScript function called `“doSayHello”`. A JavaScript function is a sequence of JavaScript statements that have been given a name. We will take a detailed look at functions in Chapter 8.

```
function doSayHello() {
  alert("Hello");
}
```

This function only performs a single action, it displays an alert that says “Hello” to the user when it is called. The line of JavaScript that displays the alert is called a *statement*. The end of a statement is marked by a semi-colon charac-ter. A function can contain many statements, each of which is ended with a ;

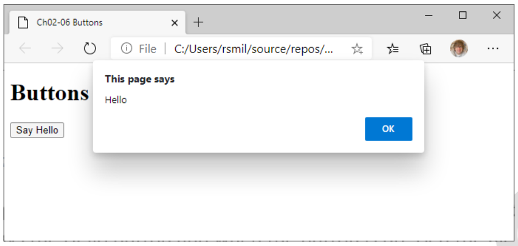
```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Ch02-06 Buttons</title>
</head>

<body>
  <h1>Buttons</h1>
  <p>
    <button onclick="doSayHello()">Say Hello</button>
  </p>
  <script>
    function doSayHello() {
      alert("Hello");
    }

  </script>
</body>
</html>
```

Ch02-06 Buttons

This is the complete HTML text of the web page. The `<script>` element is at the bottom of the body of the document. The page displays the Say Hello button and when the button is pressed the alert is displayed.



9. Ch02_fig09 Say Hello alert

Reading input from a user

```
<input type="text" id="alertText" value="Alert!">
```

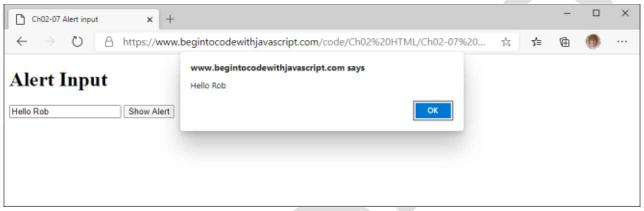
The `button` tag lets us create an element in a web page that responds to a user action. Next, we need a way of getting input from a user. The `input` tag lets us do just that. It has three attributes:

> The `type` attribute tells the browser the type of input that is being read. In the code above we are reading text, so the attribute type is set to `text`. If you set the type attribute to `password` the contents of the input are hidden as they are typed. This is how JavaScript programs read passwords in web pages.
>
> The `id` attribute gives an element a unique name. This name can be used in the JavaScript code to locate the element. If we had two input elements, we would use a different name for each. I've called the element `alertText` because this nicely reflects what the element is being used for.

The `value` attribute specifies the value in this element tag. This is how we can pre-populate an input with text. When this input element is displayed it will have the text “Alert!” in it. If we want the input to be blank when it is displayed we can set the value to an empty string.

```
<p>
  <input type="text" id="alertInputText" value="Alert!">
  <button onclick="doShowAlert()">Show Alert</button>
</p>
<script>
  function doShowAlert() {
    var element = document.getElementById("alertInputText");
    alert(element.value);
  }
</script>
```

The web page contains the input element, a button element that will call a function to display the alert and the function that uses the input text in an alert. The user can type their own text into the input and then press the Show Alert button to have the text displayed in an alert. Figure 2.10 below shows what the program looks like when it is used.


Ch02-07 Alert input
https://www.begintocodewithjavascript.com/code/Ch02%20HTML/Ch02-07%20...
Alert Input
Hello Rob
Show Alert
OK

www.begintocodewithjavascript.com says

Hello Rob

10. Ch02_fig10 Customizable alert

Ch02-07 Alert Input

If you run the example you will notice that when you press the “Show Alert” button the text you have entered in the input area is displayed in the alert.

HTML and JavaScript

It’s worth spending some time discovering how HTML and JavaScript work together, as this underpins almost all of the programs that we are going to write. The JavaScript program needs a way of interacting with the HTML document it is part of. This is interaction is provided by *methods* which are part of the *document object*. The document object is a container that holds all the elements on the page. A method is a behavior provided by an object. The document object contains methods alongside the HTML elements that make up the page. Our program uses the `getElementByID` method to get a reference to the element on the page. It then gets the text out of this element and then displays that text in an alert.

If you're not sure about this, how about an analogy. Think of the HTML document as "Rob's Car Rental". When someone comes to pick up a car they will say "I've come to pick up car registration 'ABC 123'" and I will hand them the keys and reply "It's over in bay E6". They can then go and find the car. I don't hand the customer the car over the counter (I'm not strong enough for that). I just tell them where the car is so they can go and find it.

In the case of the HTML document each of the elements in the document is like a car in the parking lot for my rental business. An element can be given an ID just like a car has a registration plate. In our document the ID is `alertInputText`. The method `getElementById` is the means by which a JavaScript program can ask the document where an element is.

```
var element = document.getElementById("alertInputText");
```

On the right-hand side of the statement above you can see the use of `getElementByID` to get the location of the text element with the id `alertInputText` The left hand side of the statement creates a *variable* to hold this location. The word `var` creates a JavaScript *variable*. A variable is a named location that stores some information that the program wants to remember.

In "Rob's Car Rental" I would offer to write down the location of a car for a customer who was afraid of forgetting where their car was. I'd give them a piece of paper with "Car Location" (the name of the "variable") and "Bay E6" (the value of the variable) written on it. In the JavaScript above the variable we are creating is called `element` (because it refers to an element in the document) and the value is the location of the text input element. This operation is called an *assignment* because the program is assigning a value to a variable. An assignment operation is denoted using the equals (=) character. We will discuss variables in detail in chapter 4. Now that the program has a variable called `element` that contains a reference to the input we can extract the text value from this element and store it in a variable called message.

```
var message = element.value;
```

The variable called `message` now contains the text that was typed into the input by the user (remember that we set this to "Alert!" in the HTML). The program can now display this text in an alert.

```
alert(message);
```

It is very important that you understand what is going on here. Up until now everything has seemed quite reasonable, and then suddenly you've been hit with something really complicated. I'm sorry about that. Just go through the code and try to map the statements back to what the program is trying to do. And remember that the equals character means “set this variable to the value”. It does not mean that the program is testing to see if one thing is equal to another.

If you are confused about how the various parts of the program fit together, consider that the program is doing exactly the same thing as if I had given a car hire customer the location of their car and then asked them to come back and tell me how much fuel there was in that car. That sequence would go as follows:

1. Get the location of the car.
2. Go to the car.
3. Get the value from the fuel level display.
4. Bring that value back to me.

In the case of the JavaScript program that is displaying the message, the sequence is:

1. Use `getElementById` method to get a reference to the input element.
2. Follow the reference to the element.
3. Get the text value from the input element.
4. Display that text in an Alert.

CODE ANALYSIS

The doShowAlert function

```
function doShowAlert() {
  var element = document.getElementById("alertInputText");
  var message = element.value;
  alert(message);
}
```

We can build on our understanding of this important aspect of JavaScript by looking at this function and considering some questions.

What would happen if you got the id of the text element wrong?

The `doShowAlert` method uses an unspoken “contract” between the HTML and the JavaScript code. The `doShowAlert` function asks the `getElementById` to find an element with the id “`alertInputText`”. If this contract is broken because the element has the name “`alertInputtext`” the `getElementById` method will not be able to deliver a result. This is a bit like me telling a car rental customer to look for their car in a location that doesn’t exist. In this case the `getElementById` method will return a special value called “null” which means “I couldn’t find anything”. This would cause the rest of the `doShowAlert` function to fail. In the case of my car rental customer they would come back and tell me that the location does not exist. In the case of the doShowAlert function there would be no error reported to the user, but the alert would not be displayed. Later in this book we'll look at how you can write code that will test for methods returning results that mean "I couldn't find what you wanted".

In JavaScript you tend to have to go hunting for the errors that you make. In some programming languages you are told about errors when they occur. In JavaScript things tend to fail silently, or just do something wrong.

What would happen if the user didn't type any text into the text area on the web page before pressing the button?

If you look at the HTML at the very start of this section you will see that the value attribute of the text tag is set to "Alert!". If the user doesn't replace this with their message the word "Alert!" will be displayed.

What would happen if I pressed the button several times?

The browser will block any activity on the web page until you clear the alert that is displayed. When you press the button again the doShowAlert function would be called again. It would make two new variables called element and message and uses them to display the appropriate message text.

What does var do?

The word var is a command to JavaScript to create a *variable*. The name of the variable follows the word var. The variable holds a value that the program wants to make use of. A program can assign values to variables by using = to tell JavaScript to perform an assignment.

Display text output

In the previous section we used a JavaScript program to read data from a web page by getting a reference to an element on the page and then reading information from that element. Displaying text on the screen is a similar process. A JavaScript program can use a reference to an object to change attributes of the element. We are going to write a program that changes the text in a paragraph into a string of text that we have entered. The complete HTML file looks like this:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Ch02-08 Paragraph Update</title>
</head>

<body>
  <h1>Paragraph Update</h1>
  <p>
    <input type="text" id="inputText" value="">[8]
```

[8] *Input text element*

```
    <button onclick="doUpdateParagraph()">Update the Paragraph</button>[9]
    <p id="outputParagraph"></p>[10]
  </p>

  <script>
    function doUpdateParagraph() {[11]
      var inputElement = document.getElementById("inputText");[12]
      var outputElement = document.getElementById("outputParagraph");[13]
      var message = inputElement.value;[14]
      outputElement.textContent = message;[15]
    }
  </script>
</body>
</html>
```

Ch02-08 Paragraph Update

This example is an extension of the previous one. Instead of displaying text using an alert this example sets the `textContent` attribute of a paragraph to the text that the user enters into the dialog box. The fundamental behavior of this program is given in these four lines.

```
var inputElement = document.getElementById("inputText");
var message = inputElement.value;
outputElement.textContent = message;
```

The first two lines set up variables that refer to the input and output elements. The third line gets the message to be displayed and the fourth puts this message onto the web page.

[9] Button that calls doUpdateParagraph

[10] Paragraph for the output

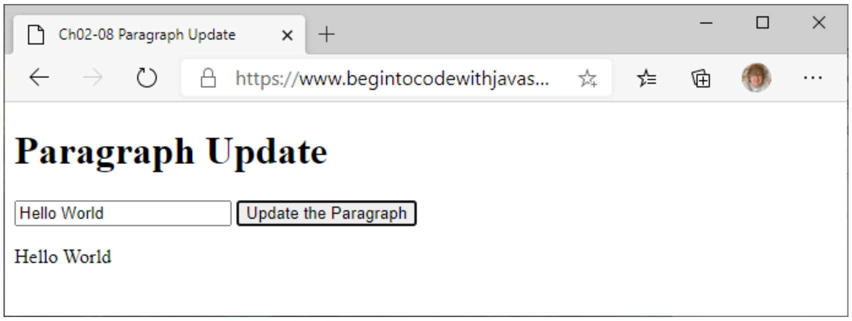
[11] Function that updates the paragraph

[12] Gets a reference to the input

[13] Gets a reference to the output

[14] Reads the text from the input

[15] Writes the text into the output



11. Ch02_fig11 Paragraph Update

MAKE SOMETHING HAPPEN

Work with object properties

You may be wondering what the `textContent` property does and how the program uses it. It might be worth investigating this. We can use the JavaScript console in the Developer Tools to do this. Find the folder on your PC that contains the sample code for the book. (If you haven't downloaded the sample code you can find the instructions in Chapter 01). Find the folder **Ch02 HTML/Ch02-08 Paragraph Update**. Double click the file **index.html** in that folder. This should open your browser and you should see a page that looks like the one in Figure 11 above. Now do the following:

Press F12 to open the Developer Tools view. Select the Console tab. Enter the following JavaScript statement:

```
var outputElement = document.getElementById("outputParagraph")
```

This is the statement in our program that gets a reference to the `outputParagraph` in the document. We now have a variable called `outputElement` that refers to the output paragraph. We can prove this by using our new variable.

```
outputElement.textContent = "fred"
```

Take a look back at the web page. You should see that the word "fred" has appeared. By setting the value of the innerText property of the paragraph we can change the text in the paragraph. A JavaScript program can read properties as well as write them. Enter the following statement:

```
alert(outputElement.textContent)
```

This will make an alert box appear with "fred" in it (because that is the `textContent` of the element referred to by `outputElement`. Now let's see what happens if we make a mistake. Try this:

```
outputElement.tetContent="test"
```

This statement looks sensible, but I've miss-typed "textContent" as "tetContent". The paragraph element does not have a tetContent property. However, this statement doesn't cause an error, but the word test is not displayed either. What happens is that JavaScript creates a new property for the outputElement variable. The new property is called "tetContent" and it is set to the value "test". You can prove this by entering the following:

```
alert(outputElement.tetContent)
```

This will display an alert showing the value in the tetContent property, which is the string "test". We will discover more about creating properties in objects in Chapter 7.

See if you can change the web page so that the name is displayed as a heading (<h1>) rather than a paragraph. You can use Visual Studio Code to edit the html for the web page. Will you have to change the JavaScript or the HTML?

Egg Timer

We now know enough to be able to create a properly useful program. We are going to create an egg timer. The user will press a button and then be told when five minutes (the perfect time for a boiled egg) have elapsed. We know how to connect a JavaScript function to an HTML button. The next thing we need to know is how to measure the passage of time. We can do this by using a JavaScript function called setTimeout. We have used functions already. The alert function accepts a string that it displays. The setTimeout function accepts two things: a function that will be called when the timer expires and the length of the timeout. The timeout length is given in thousandths of a second. The statement below will cause the function doEndTimer to run one second after setTimout was called.

```
setTimeout(doEndTimer,1000);
```

Our egg timer will use two functions. One function will run when the user presses a button to start the timer. This function will set a timer that will run the second function after five minutes. The second function will display an alert that indicates that the timer has completed.

```
function doStartTimer() {
  setTimeout(doEndTimer,5*60*1000);
}

function doEndTimer() {
  alert("Your egg is ready");
}
```

Ch02-09 Egg Timer

The `doStartTimer` function is connected to a button so that the user can start the timer. The `doEndTimer` will be called when the timer completes. I've added a calculation that works out the delay value. I want a five minute delay. There are 60 seconds in a minute and a value of 1000 would give me a one second delay. This makes it easier to change the delay. If we want to make a hard-boiled egg that takes seven minutes I just have to change the 5 to a 7. Note that the * character is used in JavaScript to mean *multiply*. You will find out more about doing calculation in Chapter 4.

MAKE SOMETHING HAPPEN

Investigate the egg timer

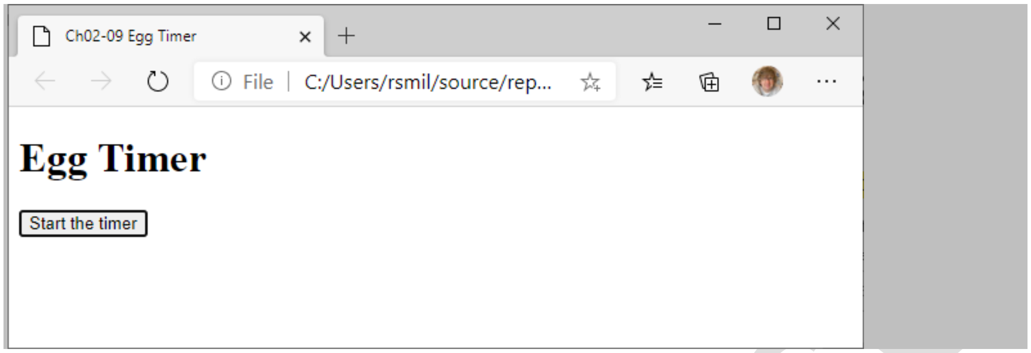
```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Ch02-09 Egg Timer</title>
</head>

<body>
  <h1>Egg Timer</h1>
  <p>
    <button onclick="doStartTimer()">Start the timer</button>
  </p>

  <script>
    function doStartTimer() {
      setTimeout(doEndTimer,5*60*1000);
    }

    function doEndTimer() {
      alert("Your egg is ready");
    }
  </script>
</body>
</html>
```

Let's take a look at how the egg timer works. Find the folder **Ch02 HTML/Ch02-09 Egg Timer**. Double click the file **index.html** in that folder to open the page.

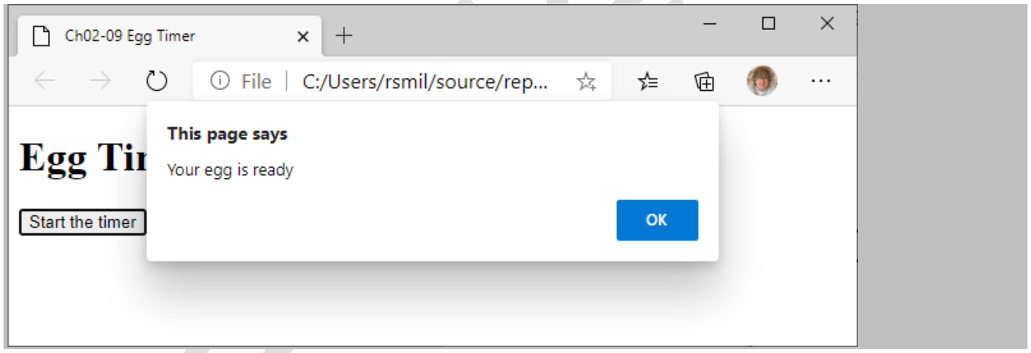


Egg Timer

Start the timer

Ch02_Readeraid_03_Fig_01 Egg timer

Click the “Start the timer” button once. This version of the code only has a delay for 10 seconds so after 10 seconds you would see the alert appear.



This page says

Your egg is ready

Ch02_Readeraid_03_Fig_02 Egg timer Ready

Click the “Start the timer” three times in succession. Wait and see what happens. Was this what you expected? It turns out that each time you press the button a new timeout is created. Now press F12 so to open up the Developer Tools. Enter the following and press Enter. What would you expect to see?

```
doEndTimer()
```

This is a call of the function that run to display the end message. You should see the alert appear telling you that your egg is ready. Click OK in the alert to close it. Enter the following and press enter. What would you expect to happen?

```
setTimeout(doEndTimer,3*1000)
```

After three seconds the alert appears, because that is the length of the timeout. You should also have seen something else appear when the function runs. You will also see an integer displayed. If you repeat the call of setTimeout you will see another value displayed. It is usually one bigger than the previous one. This number is the “id” of timer. This can be used to identify a timeout so that it can be canceled. We are not going to do that, so you can ignore this value.

See if you can change the web page so that it supports multiple cooking times. I’d like buttons for “Soft” (four minutes) “Normal” (five minutes) and “Bullet” (ten minutes). You will have to add two more buttons and two more JavaScript functions to the program.

If you want to see how I did it open up the file in **Ch02-09 Selectable Egg Timer**. My solution even displays the status of the timer.

Adding sound to the egg timer

Our egg timer works fine, but it would be nice if it could do a little more than just display an alert when our egg is ready. A web page can contain an audio element that can be used to play sounds.

```
<audio id="alarmAudio">
  <source src="everythingSound.mp3" type="audio/mpeg">
  Your browser does not support the audio element.
</audio>
```

The audio element includes another element called src that specifies where the audio data is going to come from. In this case the audio is held in an MP3 file called everythingSound.mp3 which is held on the server. The text inside the audio element is displayed if the browser does not support the audio element. I’ve given this element an id so that the code in the doEndTimer function can find the audio element and ask it to play the mp3 file.

```
function doEndTimer() {
  alarmSoundElement = document.getElementById("alarmAudio");
  alarmSoundElement.play();
}
```

Ch02-10 Alarm Egg Timer

This code looks like that in the **Ch02-08 Paragraph Update** example. In that case the getElementById method was fetching a paragraph element to be updated. In the function above getElementById is fetching an audio element to be played. An audio element provides a play method that starts it playing. The rest of the file is exactly the same as the original egg timer. If you try this program you will quite an impressive sound when your egg is ready.

Controlling Audio Playback

The egg timer page does not display anything to represent the audio element. It is “hidden” inside the HTML. You can modify an Audio element so that a player control is shown on the web page. To do this you just have to add the word `control` into the element tag:

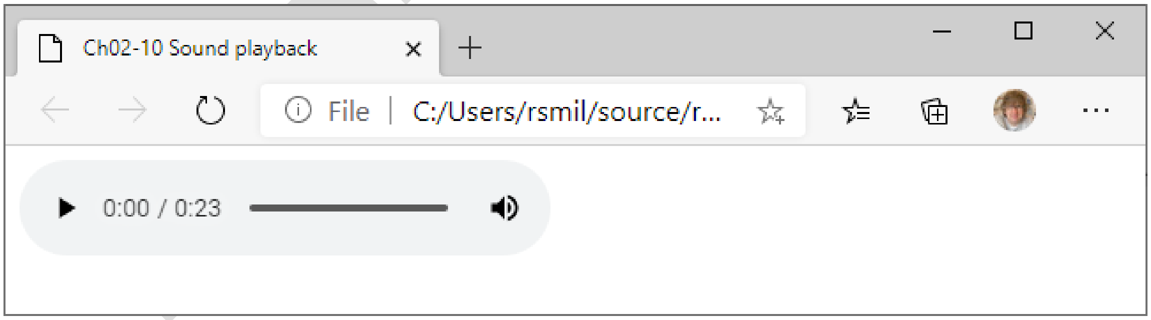
```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Ch02-11 Sound playback</title>
</head>

<body>
    <audio controls>
        <source src="everythingSound.mp3" type="audio/mpeg">
      Your browser does not support the audio element.
    </audio>
    </body>
</html>
```

Ch02-11 Sound playback

This is the complete source of an mp3 file playback page. If you visit the page you will see a simple playback control.



12. Ch02_fig12 Sound playback

This is how the playback control looks when using the Edge browser. Other browsers will look slightly different, but the fundamental controls will be the same. A viewer of the page can start the playback by pressing the play control at the far left.

An Image Display Program

The final example program in this chapter shows how JavaScript can change the content of an image displayed on the screen. You can use this technique to implement “slide shows” and also allow the user to select images for display. The image to be updated must be given an id:

```
    <img src="seaside1.JPG" alt="The image could not be found" id="pageImage"></p>
```

This `img` element displays the picture in the file seaside.JPG. A JavaScript program can change the displayed image by modifying the src attribute of the image and making it refer to a different image file:

```
var pic = document.getElementById("pageImage");
pic.src="fairground.JPG";
```

These two statements get a reference to the image and them set the src attribute of the img to refer to the image fairground.jpg. This will update the image displayed by the browser. Note that this is a repetition of a pattern that you’ve seen several times now. A program obtains a reference to a display element and then makes changes to it. You can find a complete image picker program in the example **Ch02-12 Image Picker**

MAKE SOMETHING HAPPEN

Create your own pages

You now know enough to create your own pages that contain timers, images, and buttons. Here are some ideas for you to think about:

- Make a “mood page”. The page will display buttons labelled, “Happy”, “Sad”, “Worried” etc. When the user presses a button, the page will display an appropriate message and play a piece of appropriate music.
- Make a “fitness” page. Users will press a button to select an exercise length and the page will display exercise instructions and start a timer for that exercise.
- Make a slide show. Users press a button and the page will show a sequence of images. To do this you can use a number of calls to `setTimeout` to trigger picture changes at different times in the future; perhaps one at 2 minutes, one at four minutes and so on.

What you have learned

This chapter has given you a good understanding of what the world wide web is and how it works. Here are the major points you've covered in this chapter:

- The HTTP (HyperText Transport Protocol) is used by *browsers* to request pages of data from web servers.
- Data arriving at the browser is *formatted* using HTML (HyperText Markup Language) and that a web page contains commands to the browser (for example *emphasize* this word) using tags (for example <em>) to mark out *elements* in the text. Elements can contain text, images, audio and pre-formatted text. Elements can also contain links to other web pages which can be local to a page or on distant servers.
- HTML text can contain *symbol* definitions. Symbols include characters such as '<' and '>'which are used to mark tags) and can also be used to incorporate emojis into web pages.
- An HMTL document is comprised of a line that identifies the document as HTML, followed by Header and Body elements enclosed in an HTML element. The body of the document can contain a <script> element that holds JavaScript code.
- A web page can contain a button element that runs a JavaScript function when the button is activated.
- JavaScript code interacts with the HTML document via a document object containing all of the elements of the page. The document provides methods that a program can call to interact with it. The document method getElementById can be used to obtain a reference to the page element with a particular id.
- A JavaScript program can contain variables. These are named storage locations. A variable can be assigned a value which it will store for later use. The assignment operation is denoted by the equals (=) character.
- A JavaScript function can locate elements in a document by their id attribute and then use element behaviors to change the attributes on the elements. This is how a JavaScript program could update the text in a paragraph or change the source file for an image.
- The setTimeout method can be used to call a JavaScript function at a given time in the future.

To reinforce your understanding of this chapter, you might want to consider the following "profound questions" about JavaScript, computers, programs, and programming.

What is the difference between the internet and the world wide web?

The internet is mechanism for connecting large numbers of computers together. The world wide web is just one thing that we can use the internet for. If the internet was a railway the world wide web would be one type of passenger train providing a particular service to customers.

What is the difference between HTML and HTTP?

HTTP is the Hyper Text Transport protocol. This is used to structure the conversation between a web browser and a web server. The browser uses HTTP to ask “Get me a page”. The server then gives a response, along with the page if it is found. The format of the question and the response is defined by HTTP. The design of the content of the page is expressed using HyperText Markup Language. This tells the browser things like “put a picture here” or “make this part of the text a paragraph”.

What is a url?

A url is the address of a resource that a browser wants to read. It starts with something that identifies what kind of thing is being requested. If it starts with http it means that the browser would like a web page. The middle part of the url is the network address of the server that holds the web page to be read. The final part of the url is the address on the server of the web page. This is a path to a file. If the address is omitted the server will return the contents of a file called index.html which is called the index page of a web site.

What is special about the file index.html?

The index.html file is called the *index page*. It may contain links to other web pages on the same site along with links to pages on other sites on the world wide web.

Where do I put things like image and audio files when I build a web site?

The simplest place to put images and audio files is in the same folder as the web site. So the folder that contains index.html can also contain these images and sounds. However, a path to a resource can include folders, so it is possible to organize a web site so all the images and sound files are held separately from the web pages. We will do this in the next chapter.

Why should I not use pre-formatted text for all my web pages?

The <pre>..</pre> element allows page designers to tell the browser that a block of text has already been formatted and that the browser is not to perform any additional layout. This can be useful for displaying such things as program listings which have a fixed format but it does not allow the browser to make any allowance for the target device. One of the fundamentals of web page design is that the browser should be responsible for laying out the page. The page itself should contain hints such as “take a new paragraph here” and allow the browser to sort out the final appearance.

Why should I use <em> rather than <i>?

The <i> (italic) tag means “use italic text”. The <em> tag means “make this text stand out”. If the browser is running on a device that does not support italic text it is much more useful for it to be asked to emphasize text (which it could do by changing color or inverting black and white) rather than select a character type that it is not able to display.

How are HTML tags and elements related?

A tag is the <p> marker that denotes that this text is an instruction to the browser rather than something to be displayed on a page. A complete sequence of tags (perhaps with a start and end tag) marks a complete element in a web page.

Does every HTML tag have to have a start <p> and an end </p> element?

No. Lots of tags do, for example <p> marks the start of a paragraph and </p> marks the end. But some, for example <br> (take a new line) do not.

Can you put one element inside another?

Yes. A paragraph element may contain elements of emphasized text. And an audio element contains an element that identifies the source of the audio to be played.

What is the difference between an attribute and a property?

The HTML source of a web page contains elements with *attributes*. For example <img src="seaside1.JPG"> would create an image element with a src attribute set to the image in the file “seaside1.JPG”. Within JavaScript the web page the program is part of is represented by a document object that contains a collection of objects. Each object represents one of the elements on the page. Each element object has a *property* which maps onto a particular page attribute. A JavaScript program could change the src attribute of an image element to make it display a different picture. In short; attributes are the original values that are set in the HTML and properties are the representation of these values that can be manipulated in a JavaScript program.

What is a reference?

In real life a reference can be something that you follow to get somewhere. In a JavaScript program a reference is used by a program to find a particular object. An object is a collection of data and behaviors which represent something our program is working with. JavaScript uses objects to represent elements on a web page. Each element is represented by an object. A reference is a lump of data that holds the location of a particular object.

What is the difference between a function and a method?

JavaScript contains functions which are blocks of JavaScript code that have a name. We have written functions with names like doEndTimer. Methods are functions that are held inside objects. We have used the method getElementById which is provided by the document object.