

1

The Cloud

You are reading this book because you have some programming skills, and you want to learn how to write programs for “the cloud”. If you can program you can take a problem, figure out how to solve it, and then express that solution as a set of instructions in a programming language a computer can be made to understand.

Now you want to take your problem-solving ability and use it to make “cloud based” solutions. But what is “the cloud”? What does “the cloud” do? And what kinds of problems will you have to solve if you are writing a solution that uses “the cloud”?

In this part we will start to answer these questions. We are going to take look at the origins of the cloud and identify what it is that makes an application “cloud based”. Then we’ll move on to consider how to create and manage services in the cloud. Next, we’ll address security and safety, identifying the issues to worry about and steps you can take to reduce risk when you create for the cloud. Finally, we are going to explore how the TypeScript language builds on JavaScript to make programs more secure and manageable. As we explore each topic, we’ll also take a close

look at JavaScript language features by creating some useful and fun applications.

Chapter 1

Coding for the cloud

What you will learn

In this chapter we are going to investigate the fundamentals of cloud computing and discover what makes an application “cloud based”. We are also going to start our journey with the JavaScript language by exploring how JavaScript functions allow code running in the browser to interact with the JavaScript environment. We’ll see how programs run inside a web browser and how we can interact directly with code running in the browser via the Developer Tools, which will even let us view inside our programs as they run.

I’m assuming that you are familiar with programming but just in case there are things that you don’t know (or I have a different understanding of) I’ve added a glossary at the end of this book. Whenever you see a word highlighted like this: “**computer**” it means that the word is defined in the glossary. If something doesn’t quite make sense to you, go to the glossary and check on my definition of the word that I’m using.

What is the cloud?

The internet now underpins many of our daily activities. Things like booking a table at a restaurant, buying a book, or keeping in touch with our friends are now performed using networked services. Nowadays we refer to these services as “in the cloud”. But what is the cloud? What does it do? And how can we use it? Let’s start with a look at how things were done in before we had the cloud.

The World Wide Web

The **world wide web** and the **internet** are different things. The internet was invented to make it easy to connect software together over long distances. One early “killer app” for the internet was electronic mail. Internet connected computers acting as “mail servers” managed mailboxes for users, replacing paper messages with digital ones.

The world wide web was created some years after the internet to make it easier to work with documents. Rather than having to fetch and read a paper document you use a **browser** program to load an electronic copy from a web **server**. Documents can contain links to other documents, so that you can follow a reference without having to go and fetch another physical document.

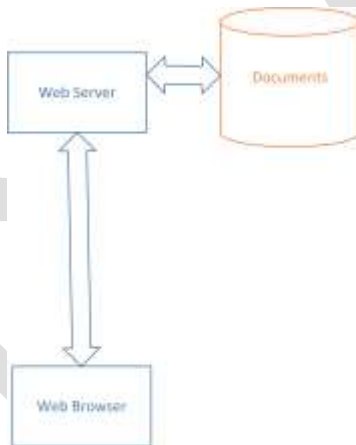


Figure 1.1 Ch01_Fig_01 Browser and Server

Figure 1.1 above shows how it works. The user sits at the browser and sends the web server computer requests for documents which are then sent back to the browser to be read. Later versions of the web added graphics so that a document could contain pictures.

Putting the web in the cloud

If you wanted a web site in the early days of the internet you would set up your own **server** computer. If your site became popular you had to increase the power of your server (or get extra ones) to handle the load. Then you might find that all your capacity was only used at times of peak demand. The rest of the time your expensive hardware was sat twiddling its digital thumbs.

The cloud addresses this problem by turning computing resources into a commodity that can be bought and sold. Rather than setting up your own server, you now rent space in the cloud and pay someone else to host your site. The amount you spend on computer resources is proportional to the demand for the service you are providing. You never have to pay for resources that you don't use. What's more, the cloud makes it possible to create and deploy new services without having to set up expensive servers to make the service available. Most of the popular cloud suppliers even have pricing plans that provide free tariffs to help you get started.

So, the server that you connect to when using a network service (including the world wide web) might be owned by the company providing the service (Facebook have invested considerable sums in setting up their own servers) but it is more likely that you will be connected to a system hosted by one of the cloud providers. Later in the book we will discover how to create an account on a cloud provider and set up a service in the cloud.

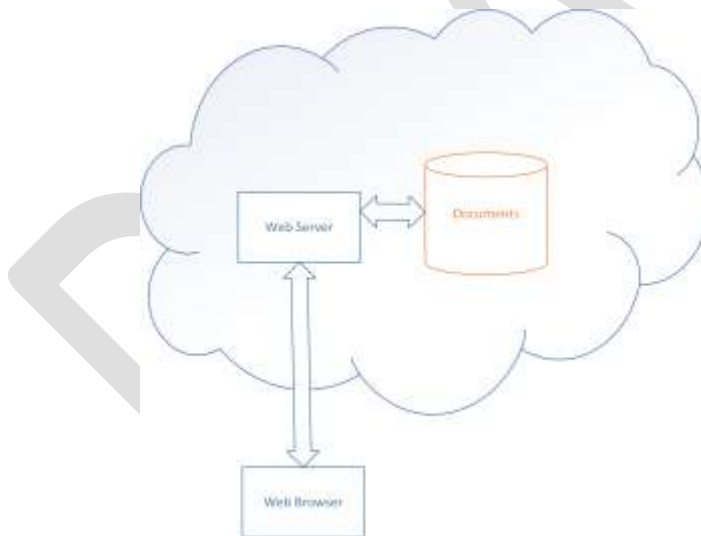


Figure 1.2 Ch01_Fig_02 HTML browser in the cloud

Figure 1.2 shows how this works. The web server and the documents are hosted in the cloud. Note that you could use a mix of different service providers, you could host the resources in one place and the server somewhere else. From the perspective of the service user, a cloud-based web site works in the same way as a server based one.



Figure 1.3 Ch01_Fig_03 Cloud application

Figure 1.3 shows the login page for a service we will be creating in Chapter 11 which connects with Internet of Things devices. My version of the service has the web address: <https://clbportal.azurewebsites.net/> If you enter that address into your browser you will be connected to a process in the cloud hosting this web site.

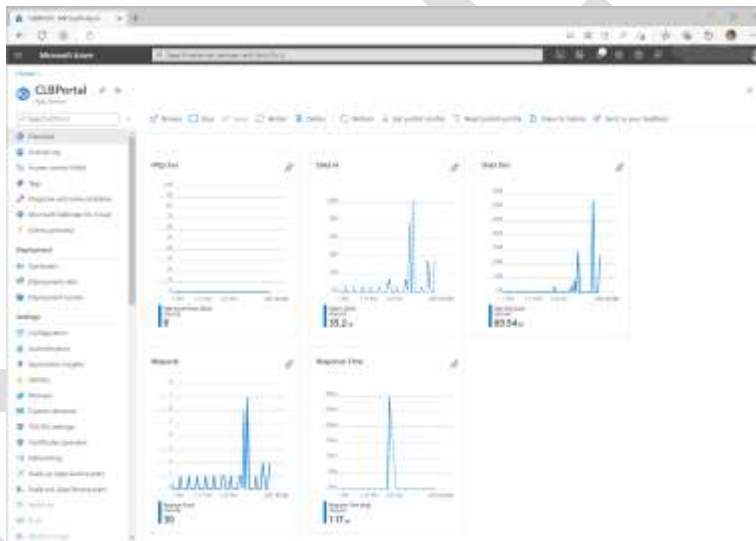


Figure 1.4 Ch01_Fig_04 Cloud management

You can manage your cloud services from the web. Figure 1.4 above shows the overview page on the Azure Portal for the service shown in Figure 1.3. It shows the amount of traffic the page is receiving and the time taken for the page to respond. There are also lots of options for service management and diagnostics. You can also use this page to increase the service provision to support many thousands of users. At the moment this service is using a free service level which supports enough users to allow demonstration and testing.

So, to answer our original questions: The **cloud** is a means by which companies can provide computing resources as a service. It hides the **physical** location of computing resources behind a **logical** address which users connect to. We can use the cloud to host our own services and make

them available for others to use via web addresses. A cloud service provider will provide a management interface for each service it hosts. Now, let's move on to take a look at the JavaScript language.

Programmer's Point

The cloud makes it a great time to be a developer

When I was learning to program it was very nearly impossible to show people what I had done. I could send them my punched cards (each card was punched with holes that contained the text of one line of my code) but it would be unlikely the program would run on the recipient's computer. Today you can invent something and make it available to the whole world by writing some JavaScript and hosting it in the cloud for free. This is tremendously empowering.

Nowadays the hard part is not making your service available but making people aware that it exists. Take a look at the Programmers Points in chapter 2, "Open-Source projects are a great place to start your career" and "GitHub is also a social network" for hints on how to do this.

JavaScript

We now know what the cloud does. It provides a means of buying (or even getting for free) space on the internet where we can host our services. JavaScript has been described as "the programming language for the cloud". Let's look at what this means.

Originally the browser just displayed information that had been received from the server. Then it was decided that it would be useful if the browser could run programs that have been loaded from web sites. Putting a program inside a web page makes the page interactive without increasing the traffic to and from the web server. The user can interact with a program running in the browser without the server having to do anything. The program in the browser can animate the display or check user input to make sure it was correct before sending it to the server.

The language developed to run inside the browser was JavaScript. There have been several different versions of the language. Early ones suffered from a lack of standardization. Browsers from different companies provided different sets of features that were accessed in different ways. But this has now settled down. The specification of the language is now based on a worldwide standard managed by a standards organization called ECMA. We are going to be using version ES6 of the language.

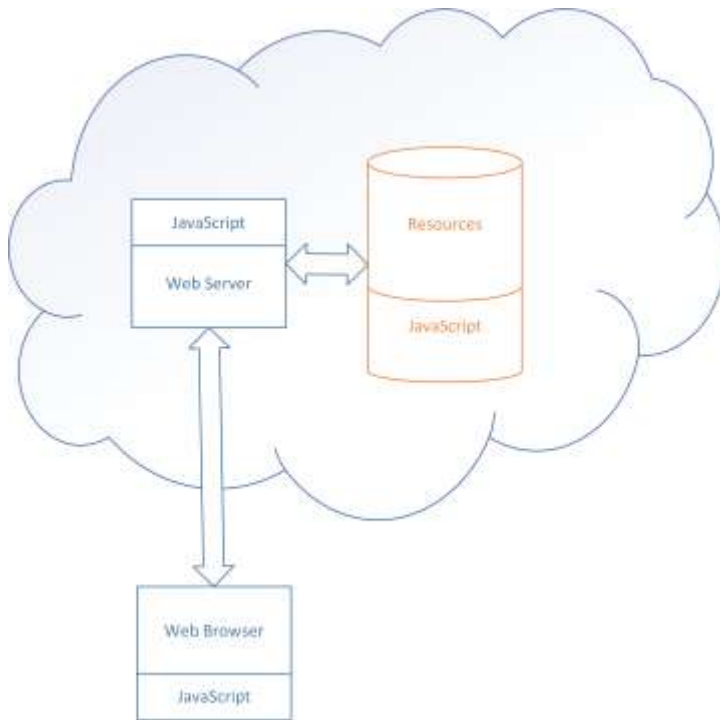


Figure 1.5 Ch01_Fig_05 JavaScript powered web

Figure 1.5 shows how a modern, JavaScript enabled browser and server work together. The web pages and the server program both contain JavaScript elements and the document store has been replaced with a range of resources which can also contain JavaScript code.

JavaScript has become extremely popular. Whenever you visit a website, you will almost certainly be running JavaScript code. JavaScript has also become very popular on web servers (the machines on the internet that deliver the information the browser requests). A technology called “**node.js**” allows JavaScript programs to run on a server to respond to requests from browsers. The web page shown in Figure 1.3 is produced by a JavaScript program running in the cloud using node.js. We’ll discover how to do this later, for the rest of this chapter we are going to be running JavaScript in the web browser. We’ll start by looking at a JavaScript hero, the JavaScript Function.

JavaScript Heroes: functions

This is the first of our “JavaScript heroes”. JavaScript heroes are features of the language which make it super useful for creating cloud applications. A cloud application is all about events. Events happen when the user clicks the mouse button, when a message arrives from a server and when a timer goes tick. In all these situations you need a way of connecting a behavior to what

has just happened. As we shall see, functions in JavaScript simplify the process of connecting code to events. You might think you already know about functions in a programming language. However, I'd advise you to work through this section very carefully anyway. I'm sure that there will be at least one thing here that you didn't know about functions in JavaScript. This is because JavaScript has a very interesting implementation of the function which sets it apart from other programming languages. Let's start with an overview of functions and then drill down into what makes them special in JavaScript.

The JavaScript function object

A JavaScript function is an object that contains a "body" made up of JavaScript statements that are performed when the function is called. A function object contains a name property which gives the name of that function. Functions can be called by their name, at which point the program execution is transferred into the statements that make up the function body. When the function completes the execution returns to statement after the one called the function. Functions can be made to accept values to work on and a function can also return a result value.

```
function doAddition(p1, p2) {  
    let result = p1 + p2;  
    alert("Result:" + result);  
}
```

The code above defines a function with the name `doAddition`. The definition is made up of the header (the part with the name of the function and the parameters it accepts) and the body (the block of two statements that are obeyed when the function is called). The function above calculates the sum of two values and displays the result in an alert box. The `alert` function is one of many "built-in" functions that are provided by the browser Application Programming Interface or API. Learning how to use the facilities provided by the API in a system is a huge part of learning how to be an effective developer.

Programmer's Point

You don't need to know about every JavaScript API function

There are thousands of different functions available to a JavaScript program. The `alert` function is one of them. You might think you need to know about all of them, but actually you don't. I don't know anyone who knows all the functions available to JavaScript programs. But I know lots of people who know how to use search engines to find things when they need them. Don't be afraid to look things up, and don't feel bad about not knowing everything.


```
doAddition(3,4);
```

We call the `doAddition` function as shown above. The values 3 and 4 are called the **arguments**. Argument values are mapped onto the parameters in the function. When the above call of `doAddition` starts to run the parameter `p1` will hold the value 3 and the parameter `p2` will hold the value 4. This leads to the display of an alert box that displays the value 7.



Figure 1.6 Alert box

Figure 1.6 above shows the alert box that is displayed when the function runs. An alert box always has the name of the originator at the top. In this case the function was running in a page on a website located at `beginintocloud.com`. The `alert` function is the first JavaScript function that we have seen. It asks the browser to display a message and then waits for the to click the OK button. From an API point of view, we can say that the `alert` function accepts a string of text and displays it.

Lifting the lid on JavaScript

Wouldn't it be nice if we could watch the `doAddition` function run? It turns out that we can. Modern browsers contain a "Developer Tools Console" which lets you type in JavaScript statements and view the results. How you start the console depends on the browser you are using:

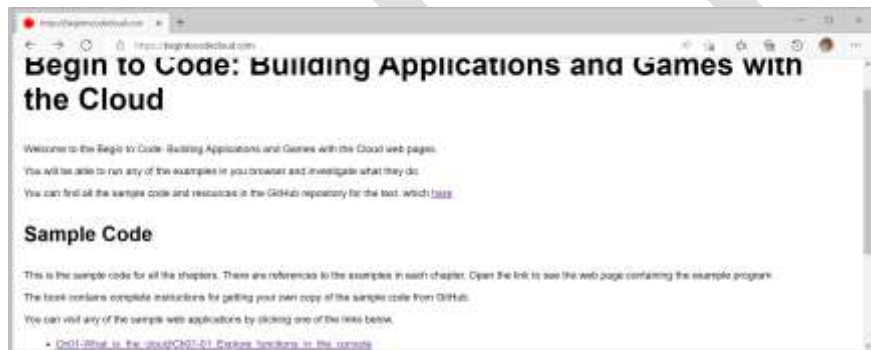
Operating System	Browser	Sequence	Notes
Windows	Edge	F12 or CTRL+SHIFT+J	The first time you do this you will be asked to confirm the action.
Windows	Chrome	F12 or CTRL+SHIFT+J	
Windows	FireFox	F12 or CTRL+SHIFT+J	
Windows	Opera	CTRL+SHIFT+J	
Macintosh	Safari	CMD+OPTION+C	You need to go to Preferences->Advanced->Show Develop Menu and select "Show Develop Menu" to enable it.
Macintosh	Edge	F12 or CTRL+SHIFT+J	
Macintosh	Chrome	CMD+OPTION+J	
Macintosh	FireFox	CMD+SHIFT+J	
Macintosh	Opera	CMD+SHIFT+J	This also works with the Chromium browser on the
Linux	Chrome	F12 or CTRL+SHIFT+J	

The table above gives the shortcut keys for different browsers and operating systems. Note that the console will look slightly different on each browser, but the views that we are going to use are present on all of them. Let's do our first "Make Something Happen" and use the Developer Tools console to explore functions from our web browser.

Make Something Happen

Explore functions in the console

All the example code for this book is available in the cloud. Of course. You can find the sample pages at begintocodecloud.com. Open this web site and scroll down to the Samples section on the page.



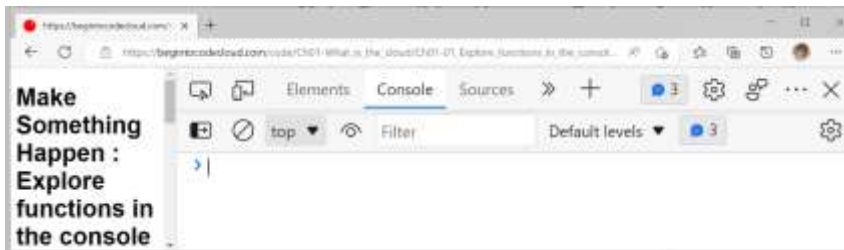
Ch01_inset01_01 Web Site Samples

The sample code is presented as a list of links. There is a link for each sample page. We are going to do the very first sample so click **Ch01-What_is_the_cloud/Ch01-01_Explore_functions_in_the_console**.



Ch01_inset01_02 Explore Functions

This is the web page for this exercise. Press the key sequence to open the Developer Tools in your browser. The screenshots for this section are from the Edge browser running in Windows.



Ch01_inset01_03 Developer Console

The Developer Tools will open on the right-hand side of your page. You can make the tools area larger by clicking the line separating the sample code from the tools and dragging it to the left as shown above. The web page will automatically resize. The Developer Tools contain several different tabs. We will look at the Elements and Sources tabs later. For now, click the Console tab to open the console.

We can type JavaScript statements into the console, and they will be performed in the browser and the results displayed. The console provides a '>>' prompt which is where you type commands. Click the console and start typing "doAddition" and watch what happens.



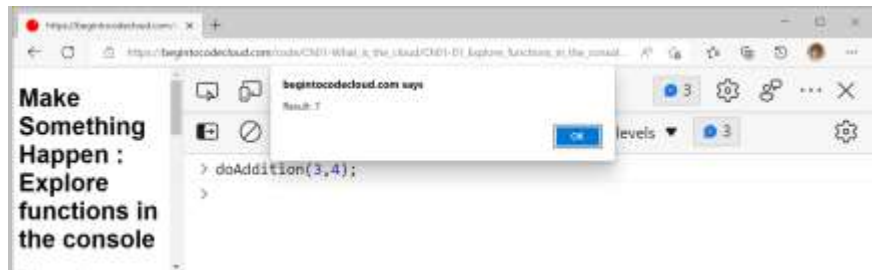
Ch01_inset01_04 Typing doAddition

As you type the text the console presents you with a menu of things you could type in. This is a very useful feature. It saves you typing, and it makes it less likely that you will type something incorrectly. You can move up and down the menu of items by using the arrow keys or the mouse. Click **doAddition** in the list or press the TAB key when it is selected. Now fill in the rest of the function call by adding 3 and 4, the two arguments:



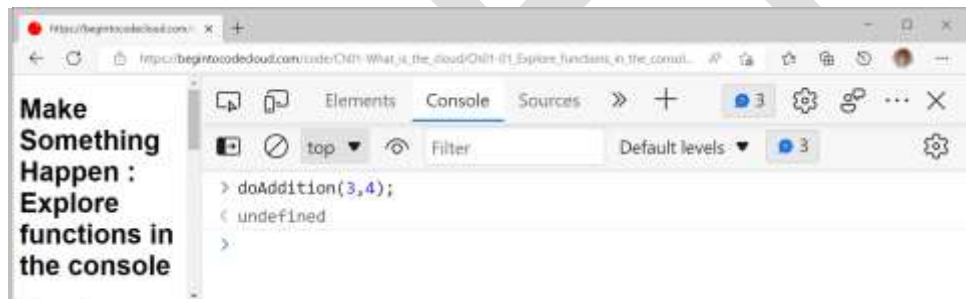
Ch01_inset01_05 Calling doAddition

When you have finished the console should look like the above. Now press enter to run the function.



Ch01_inset01_06 doAddtion alert

The `alert` function has control and is displaying the alert box containing the result. Note that you can't enter new statements into the console at this point. The only thing you can do next is press the OK button in the alert to clear it. Do this.



Ch01_inset01_07 doAddtion completed

When you click on OK the alert box disappears and the `doAddition` function completes. You can now enter further commands in the console.

CODE ANALYSIS

Calling functions

A code analysis section is where we examine something that we have just seen and answer some questions you might have about it. There are a few questions you might have about calling functions in JavaScript. Keep the browser open and displaying the console so you can find out the answers.

Question: What does the `undefined` message mean in the console after the call of `doAddition`?

The console takes a JavaScript statement, executes it, and displays the value generated by the statement. If the statement calculates a result the statement will have the value of that

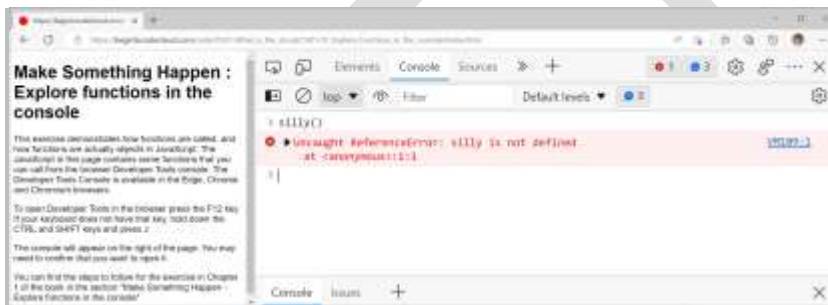
result. This means you can use the console as a calculator. If you type in 2+2 the console will display 4.

```
> 2+2  
< 4
```

The expression 2+2 is a valid JavaScript statement that returns the value of the calculation result. So the console displays 4.

However, the `doAddition` function does not deliver a result. It has no value to return, so it returns a special JavaScript value called **undefined**. Later we will discover JavaScript functions that do return a value.

Question: What happens if I try to call a function that is not available?



Ch01_inset02_01 calling missing function

Above you can see what happened when I tried to call a function called “silly”. JavaScript told me that the function has not been defined.

Question: What would happen if I added two strings together?

In JavaScript you can express a **string** of text by enclosing it in double quotes or single quotes. We will discuss JavaScript in detail strings later in the text.

```
> doAddition("hello", "world");
```

The call of `doAddition` above has two string arguments. When the function runs the value of `p1` is set to “hello” and the value of `p2` is set to “world”. The function applies the + operator between the parameters to get the result.

```
let result = p1 + p2;
```

Above you can see the statement in the `doAddition` function that calculates the value of the result of the function. The statement defines a variable called `result` which is then set to sum of the two parameters. We will be looking at the **let** keyword later in the book (or you can look up **let** in the Glossary). The JavaScript selects a + operator to use according to the **context** of the addition. If `p1` and `p2` are numbers JavaScript will use the numeric version of +.

If `p1` and `p2` are strings of text JavaScript will use the string version of `+` and set the value of `result` to a string containing the text "helloworld". You might find it interesting to try adding strings to numbers and watching what JavaScript does. If you look in the `doAddition` function itself you will find a statement does this.

Question: What happens if I subtract one string from another?

Adding two strings together makes sense but subtracting one string from another is not sensible. We can investigate what happens if we do this because the web page for this exercise contains a function called `doSubtraction`. Normally you would give this function numeric arguments. Let's discover what happens if we use text.

```
> doSubtraction("hello", "world");
```

If you make the above call of `doSubtraction` you get the following message displayed by the alert:

```
Result: NaN
```

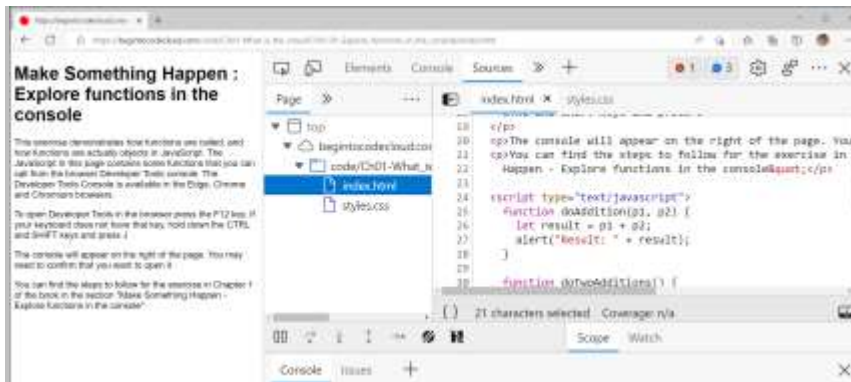
The value "NaN" means "not a number". There is only one version of the `-` operator, the numeric one. However, this can't produce a number as a result because it is meaningless to subtract one string from another. So the result of the operation is to set the value of `result` to a "special" value **NaN** to indicate that the result is not a number. Later we'll discover more about the special values in JavaScript programs.

Question: Why do some strings have `"` around them and some have `'`.

When the debug console shows you a string value it will enclose the string in single quote characters. However, in some parts of the program strings are delimited by double quote characters. In JavaScript you can use either double or single quotes to mark the start and end of a string.

Question: Where do these functions come from?

That's a good question. The function statements are in the web page loaded by the browser from the `begintocloud.com` server. We can use the Developer Tools to view this file. We must change the view from Console to Source to do this. Click the Sources tab which is next to the Console tab on the top row.

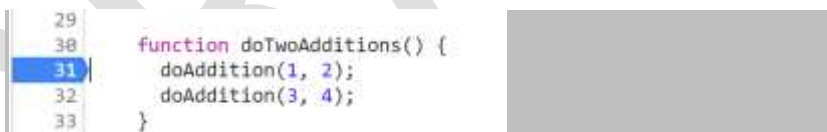


Ch01_inset02_02 viewing the web page source

The Sources view shows you all the files behind the website that you are visiting. This site has two files; a styles.css file which contains style definitions (of which more in the next chapter) and an index.html file which contains the text of the web page, along with the JavaScript programs. If you select the index.html file as shown above, you will see the contents of the file including the JavaScript for `doAddition`. There is another function called `doTwoAdditions` which calls `doAddition` twice.

Question: Can we watch the JavaScript run?

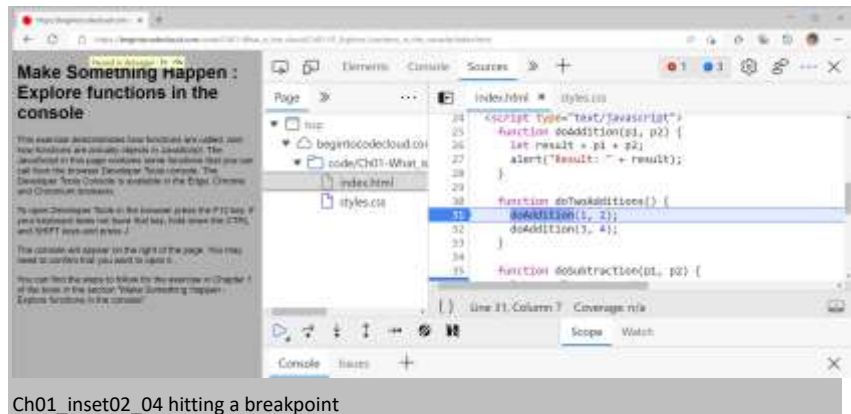
Yes. We can set a “breakpoint” at a statement and when that statement is reached the program will pause and we can go through it one step at a time. This is a wonderful way to see what a program is doing. Put a breakpoint at the first statement of `doTwoAdditions` by clicking in the left margin to the left of the line number:



Ch01_inset02_04 setting a breakpoint

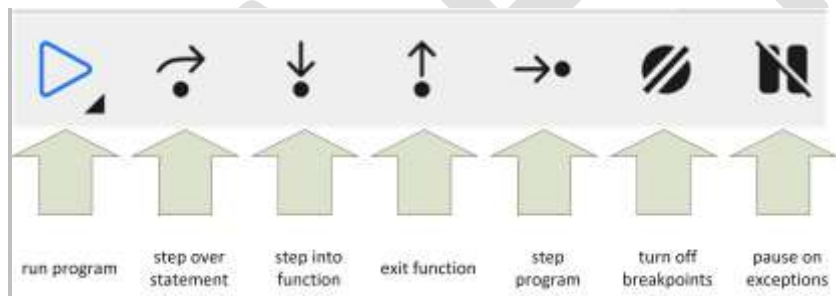
The breakpoint is indicated by the arrow highlighting the line number. It will cause the program to pause when it reaches line 31 in the program. Now we need to call the `doTwoAdditions` function. Select the Console tab, type in the following and press enter:

```
> doTwoAdditions();
```



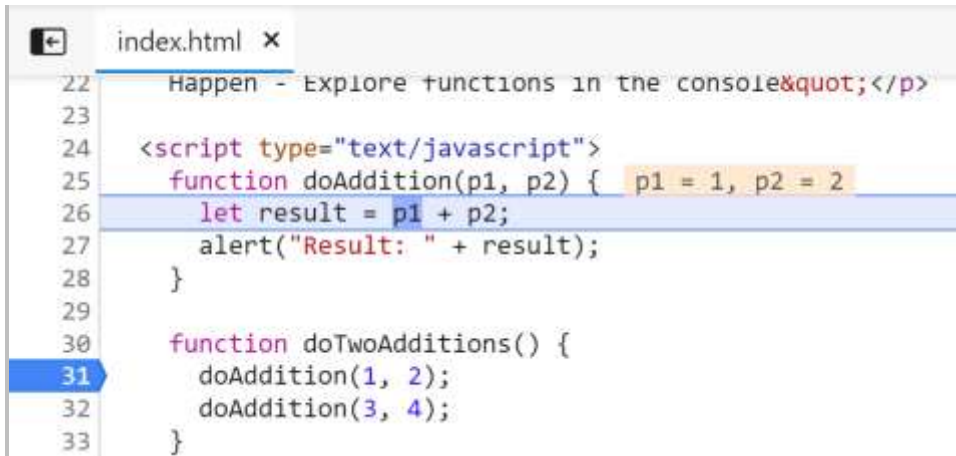
Ch01_inset02_04 hitting a breakpoint

Above you can see what happens when a breakpoint is hit. The browser shows the program paused at the statement with the breakpoint. The most interesting part of this view is the control buttons you can see towards the bottom of the page:



Ch01_inset02_05 program controls

These controls might look a bit like cave paintings, but they are very useful. They control how the browser will work through your program. The one that we will use first is “step into function” (the one third from the left). Each time you press this control the browser will perform one statement in your program. If the statement is a function call the browser will step into the function. You can use the “step over statement” control to step over function calls and the “exit function” to leave a function that you have just entered. Press “step into function”.

A screenshot of a web browser's developer console. The top bar shows 'index.html' with a close button. The console displays the following code:

```
22   Happen - Explore functions in the console"</p>
23
24   <script type="text/javascript">
25     function doAddition(p1, p2) { p1 = 1, p2 = 2
26       let result = p1 + p2;
27       alert("Result: " + result);
28     }
29
30     function doTwoAdditions() {
31       doAddition(1, 2);
32       doAddition(3, 4);
33     }
```

Line 31 is highlighted with a blue bar, indicating a breakpoint. The code defines two functions: `doAddition` and `doTwoAdditions`. The `doAddition` function takes two parameters, `p1` and `p2`, and returns their sum. The `doTwoAdditions` function calls `doAddition` twice with different arguments.

Ch01_inset02_06 viewing program execution

The highlighted line has now moved to the first statement of the `doAddition` function. The debugger shows you the values in the parameters. If you keep pressing the “step into function” button in the control panel you can see each statement obeyed in turn. Note that you will have to click the OK button in the alert when you perform the statement on line 27 that calls `alert`. If you get bored, you can press the “run program” button at the left of the program controls to run the program. You can clear the breakpoint at line 31 by clicking it.

You can add as many breakpoints as you like and you can use this technique on any web page that you visit. It is interesting to see just how much complexity that there is behind a simple site. Leave the browser open at this page, we will be making some more things happen in the following text.

References to JavaScript function objects

We have seen that function in a JavaScript program is function is represented by a JavaScript function object. A JavaScript function **object** is managed by **reference**. Our programs can contain reference variables that can be made to refer functions.

```
function doAddition(p1, p2) {
  let result = p1 + p2;
  alert("Result:" + result);
}
```

We’ve seen the definition above before. It defines a function called `doAddition` that adds two parameters together and display the result. When JavaScript sees this it creates a function object to

represent the function and creates a variable called `doAddition` that refers to the function object.

```
doAddition(1,2);
```

The statement above calls the `doAddition` function which will display an alert containing a result of 3. JavaScript variables can contain references to objects so you can write statements like this in your program:

```
let x = doAddition;
```

This statement creates a variable called `x` and makes it refer to the same object as the `doAddition` function.

```
x(5,6);
```

This statement calls whatever `x` is referring to and passes it the arguments 5 and 6. This would call the same object that `doAddition` is referring to (because that is what `x` is referring to) resulting in an `alert` with the message "Result:11" being displayed. We can make the variable `x` refer to a different function.

```
x = doSubtraction;
```

The above statement only works if we have previously declared a function called `doSubtraction` which performs subtraction. The statement makes `x` refer to this function.

```
x(5,6);
```

When the statement above calls `x` it will run the `doSubtraction` function and display a result value of -1, because that is the result of subtracting 6 from 5. We can do evil things with function references. Consider the following statement:

```
doAddition = doSubtraction;
```

If you understand how evil this statement is, you can call yourself a “function reference ninja”. It is completely legal JavaScript that means that from now on a call of `doAddition` will now run the `doSubtraction` function.

Function expressions

You can create JavaScript functions in places where you might not expect to be able to. We are used to setting variables by assigning **expressions** to them:

```
let result=p1+p2;
```

The above statement assigns the expression `p1+p2` to a variable called `result`. However, you can also assign a variable to a *function expression*:

```
let codeFunc = function (p1,p2){ let result=p1+p2; alert("Result: "+result);};
```

The above statement creates a function object which does the same as the `doAddition` function we have been using. I’ve put the entire function on a single line but the statements are exactly the same. The function is referred to by a variable called `codeFunc`. We can call `codeFunc` in the same way as we used `doAddition`. The statement below calls the new function and would display a result of 17

```
codeFunc(10,7);
```

Function references as function arguments

This is probably the most confusing section title so far. Sorry about that. What we want to do is look at how you can pass function references into functions. In other words, a program can tell a function which function to call. Later in this chapter we are going to tell a timer which function to call when the timer goes tick. This is how it works.

An **argument** is something that which is passed into a function when it is called. We have passed two arguments (**p1** and **p2**) into the **doAddition** function each time we call it (these are the items to be added). We can also use references to functions as arguments to function calls.

```
function doFunctionCall(functionToCall, p1, p2){  
    functionToCall(p1,p2);  
}
```

The function **doFunctionCall** above has three parameters. The first (**functionToCall**) is a function to call, the second (**p1**) and third (**p2**) are values to be passed into that function when it is called. All that **doFunctionCall** does is call the supplied function with the given arguments. It's not particularly useful, but it does show that you can make a function that accepts function a reference as a parameter. We could call **doFunctionCall** like this:

```
doFunctionCall(doAddition,1,7);
```

This statement calls the **codeFunc** function. The first argument to the call is a reference to **doAddition**, the second argument is 1 and the third 7. The result would be an alert that displayed "Result:8". We can get different behaviors by using **doFunctionCall** to call different functions.

We can take this further and use function expressions as arguments to function calls as shown in the statement below which defines a function which used as an argument in a call to the **doFunctionCall** function. A function created as an argument has no name and so it is called an anonymous function.

```
doFunctionCall(function (p1,p2){ let result=p1+p2; alert("Result: "+result);},1,7);
```

Make Something Happen

Fun with Function Objects

Function objects can be confusing. Let's use our debugging skills to take a closer look at how they work. If you have left the browser at the page for the previous "Make Something Happen" just continue with that. Otherwise need to go to the book web page at

beginnertocloud.com and then select the sample **Ch01-What_is_the_cloud/Ch01-01_Explore_functions_in_the_console**. Then press function key F12 (or CTRL+SHIFT J) to open Developer Tools and select the console.

```
> let x = doAddition;
```

Type the above into the console. Note that we don't put any arguments on the end of [doAddition](#) because we are not calling it, we are specifying the name of the reference to the function. Now press enter.

```
> let x = doAddition;  
< undefined
```

The console shows the value "undefined" because the console always displays the value returned by a statement and the act of assignment (which is what the program is doing) does not return a value. After this statement has been performed the variable **x** now refers to the [doAddition](#) function. We can check this by looking at the **name property** of **x**. A function object has a name property which is the name of the function. Type in "x.name;" press enter and look at what comes back:

```
> x.name;  
< 'doAddition'
```

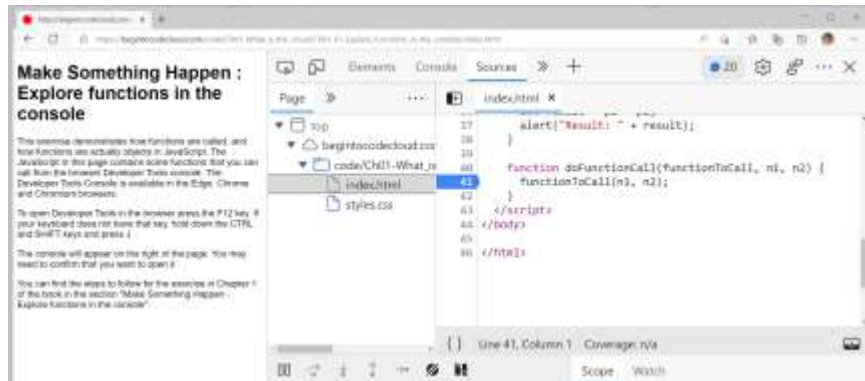
The console displays the value returned by the statement. In this case the statement is accessing the **name** property of the variable **x** which is the string `doAddition`.

```
> x.name;  
< 'doAddition'
```

Now let's call **x** with some arguments Type in the following statement and press Enter.

```
> x(10,11);
```

Since **x** refers to [doAddition](#), you will see an alert displaying the value 21. Next, we are going to feed the **x** function reference into a call of [doFunctionCall](#), but before we do that we are going to set a breakpoint so that we can watch the program run. Select the Sources tab and scroll down the `index.htm` source file until you find the definition of [doFunctionCall](#). Click the left margin near the line number 41 to set a breakpoint inside the function at statement 41.

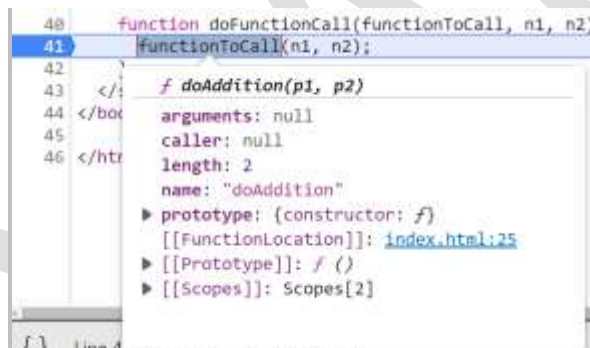


Ch01_inset03_01 breakpoint in doFunctionCall

Now return to the Console view, type the following statement and press Enter.

```
> doFunctionCall(x,11,12);
```

When you press Enter the console calls `doFunctionCall`. When it reaches the first statement in the function it hits the breakpoint and pauses.



Ch01_inset03_02 program paused in doFunctionCall

Above you can see the program paused. The `doFunctionCall` function has been entered and the parameters to the function have been set to the arguments that were supplied. If you hover the mouse pointer over `functionToCall` in line 41 you will see a full description of the value in the parameter. The description shows that the parameter refers to the `doAddition` function.

Repeatedly press the “step into function” button to watch the program go into the `doAddition` function, calculate the result and display the result in an alert. Then click OK in the alert to clear it and press the clear the alert and press the “Run Program” button (it’s the one on the left) to complete this call. Finally, return to the console for the “grand finale” of this Make Something Happen.

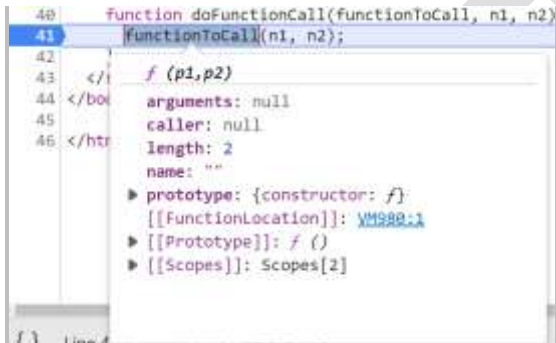
In the “grand finale” we are going to create an anonymous function and pass it into a call of

`doFunctionCall`. The function will be defined as an argument to `doFunctionCall`. The statement we are going to type is a bit long and you must get it exactly right for it to work. The good news is that the console will suggest sensible things to type. If you get an error, you can use the up arrow key to get the line back so that you can edit it and fix any mistakes.

```
> doFunctionCall(function (p1,p2){ let result=p1+p2; alert("Result: "+result);},1,7);
```

Ch01_inset03_03 calling anonymous function

Now press enter to execute this statement. The program will hit the same breakpoint as before, but the display will be different:



Ch01_inset03_04 anon function in doFunctionCall

This time the name property of the function is an empty string. The function is anonymous. Press the “step into function” button to see what happens when an anonymous function is called.



Ch01_inset03_05 stepping through an anon function in doFunctionCall

The browser has created a temporary file to hold the anonymous function while it is in the debugger. The file is called VM980. When you do this exercise, you might see a different name. We can step through the statements in this file using “step into function”. If you want to just run the function to completion you can press the “Run program” button in the program controls.

Anonymous functions are used a lot in JavaScript, particularly when calling API functions to perform tasks. An object from the JavaScript API will signal that something has happened by calling a function. The quickest and most convenient way to create the function to be called is to declare it as an anonymous function.

Returning values from function calls

Up until now we have just called functions and they have done things. They have not returned a

value. They have returned the value undefined. Now we are going to investigate how a function can return a value and how a program can use the value that is returned.

```
function doAddSum(p1, p2) {  
    let result = p1 + p2;  
    return result;  
}
```

The function `doAddSum` above shows how a function can return a value. The `return` keyword is followed by an expression that gives the value to be returned when the function is called.

```
let v = doAddSum(4,5);
```

The statement above creates a variable called `v` and set the value of this variable to the result of the call of `doAddSum` – in this case the value 9 (the result of adding 4 to 5). The return from a function can be used anywhere you can use a value in a function.

```
let v = doAddSum(4,5) + doAddSum(6,7);
```

In the statement above the `doAddSum` function would be called twice and the value of `v` would be set to 22. We can also use function returns as arguments in function calls.

```
let v = doAddSum(doAddSum(4,5), doAddSum(6,7));
```

The code above looks a bit confusing but JavaScript would not have a problem performing it. The outer call of `doAddSum` would be called first, and then the two further calls would run to calculate the values of the two arguments. Then the outer call would run with these values. Note that the above statement is not an example of **recursion** (where a function calls itself). It shows how function calls can be used to produce values to be used as arguments to function calls.

A function can contain multiple `return` statements so it can return from different places in the function code. You can also use a `return` from a function to “escape” from inside the middle of deeply nested loops. However, if you use `return` like this you might end up making code that is

harder to debug.

Programmer's Point

Try to design your code to make it easy to debug and maintain

You will spend at least as much time debugging and maintaining code as you will writing it. Worse still, you will frequently be called on to debug and maintain programs that have been written by other people. Even worse still, six months after you've written a piece of code you become one of the "other people". I've occasionally asked myself "What idiot wrote this code?" only to find out that it was me.

When you write a program, try to make sure that it is going to be easy to debug. Consider the implementation of `doAddSum` below.

```
function doAddSum(p1, p2) {  
  let result = p1 + p2;  
  return result;  
}
```

You might think that it would be more efficient to return the result directly and get rid of the `result` variable:

```
function doAddSum(p1, p2) {  
  return p1 + p2;  
}
```

The above version of the function works fine. And it might even save a few millionths of a second when it runs (although I doubt this because browsers are very good at optimizing code). However, the second one will be harder to debug because you can't easily view the value of the result that it returns. With the original code I can just look at the contents of the `result` variable. In the "improved" version I'll have to mess around a bit to find out what value is being returned to the caller.

It's a similar issue with lots of `return` statements in a function. If the function containing lots of returns delivers the wrong result you have to step through it to work out the route it is following to deliver that particular result. If the function only has one return statement you know exactly where the result is coming from.

If the function just performs a task a good trick is to make a function return a status code so that the caller knows exactly what has happened. I often use the convention that an empty string means that the operation worked, where as a string contains a reason why it failed.

If the function is supposed to return a value you can use the JavaScript values `null` and `undefined` to indicate that something has not worked

Oh, and if you find yourself thinking “What idiot wrote this code?”, don’t be so hard on the “idiot”. They were probably in a hurry, or lacked your experience or maybe, just maybe, there might a reason why they did it that way that you don’t know.

Returning multiple values from a function call

A problem with functions is that they can only return one value. However, sometimes we would like a function that returns multiple values. Perhaps we need a function to read information about a user. The function returns a name and an address along with status which indicates whether or not the function has succeeded. If the status is an empty string it means that the function worked. Otherwise, the status string contains an error message.

```
function readPerson() {  
    let name = "Rob Miles";  
    let address = "House of Rob in the city of Hull";  
    let status = "";  
}
```

Above is an implementation of a `readPerson` function that sets up some return values but doesn’t return anything. What we want now is a way the function can return these values to a caller.

Returning an array from a function call

```
function readPersonArray() {  
    let name = "Rob Miles";  
    let address = "House of Rob in the city of Hull";  
    let status = "";  
    return [status, name, address];  
}
```

The above code creates a function called `readPersonArray` that returns an array containing the status, name and address values. We create an array in JavaScript by enclosing a list of values in brackets.

```
let reply = readPersonArray();
```

The statement above shows how we would create a [reply](#) variable that holds the result of a call to [readPersonArray](#). We can now work with the values in the array by using an index value to specify which element we want to use in our program.

```
let status = reply[0];
if(status != "") {
    alert("Person read failed:" + status);
}
```

The above code puts the element at the start of the reply array (JavaScript arrays are indexed starting at 0) into a variable called [status](#). This should be the status value of the call that has just been made. If this element is not an empty string the code displays an alert containing the status so that the user can see that something has gone wrong. If you look at the code for [readPersonArray](#) you'll see that the element at the start of the array is the status value, so this code would display an alert if the [status](#) variable contains an error message.

This code works, but it has an obvious disadvantage. The person making the call of [readPersonArray](#) needs to know the order of the values returned by the function. For this reason, I don't think using an array here is a very good idea. Let's look at a better one.

Programmer's Point

Use extra variables to make code clearer

You might look at the code above and decide that I've used a variable when I don't need to. I've created a variable called [status](#) that contains a copy of the value in [reply\[0\]](#). I've done this because it makes the code that follows much clearer. The test of the status and the display in the alert make a lot more sense to the reader than they would if the code contained the variable [reply\[0\]](#). This won't slow the program down or make it larger because the JavaScript engine is very good at optimizing statements like these.

Returning an object from a function call

```
function readPersonObject() {
    let name = "Rob Miles";
    let address = "House of Rob in the city of Hull";
    let status = "";
    return {status:status, name:name, address:address};
}
```

The above code creates a function called `readPersonObject` that returns an object containing status, name and address properties.

```
let reply = readPersonObject();
if(reply.status != "") {
    alert(reply.status);
}
```

The code above shows how the `readPersonObject` function would be called and the status property tested and displayed. Note that this time we can specify the parts of the reply that we want by using their name. If you want to experiment with these functions you can find them in the example web page we have been using for this chapter: **Ch01-What_is_the_cloud/Ch01-01_Explore_functions_in_the_console**

Programmer's Point

Make good use of object literals

This way of creating objects in JavaScript programs is called an object literal. I'm a big fan of them. They are a great way to create data structures which are easy to use and understand, right at the point you want to use them. You can also make an object literal to supply as an argument to a function call. If I want to supply name and address values to a function, I can do this because a function can have multiple arguments.

```
function displayPersonDetails(name, address) {
    // do something with the name and address here
}
```

The function `displayPersonDetails` has two parameters so that it can accept the incoming information. However, I'd have to be careful when calling the function because I don't want to get the arguments the wrong way round:

```
displayPersonDetails("House of Rob", "Rob Miles");
```

This would display the details of someone called "House of Rob" living at "Rob Miles". A much better way would be to have the function accept an object which contains name and address properties:

```
function displayPersonDetails(person) {
    // do something with the person.name and person.address here
}
```

When I call the function I create an object literal to deliver the parameters.

```
displayPersonDetails({ address:"House of Rob", name:"Rob Miles"});
```

This call of the function creates an argument which is an object literal containing the name and address information for the person. It is now impossible to get the properties the wrong way round in the function.

Make a Console Clock

We are now going to apply what we have learned to create a clock that we can start from the console. The clock will display the hours, minutes and seconds on a web page. At the moment we don't know how to display things on web pages (that is a topic for Chapter 2) so I've provided a helper function that we can use. You can use the Developer Tools to see how it works.

Getting the date and time

Our clock is going to need to know the date and time so that it can display it. The JavaScript environment provides a [Date](#) object we can use to do this. When a program makes a new [Date](#) object the object is set to the current date and time.

```
let currentDate = new Date();
```

The statement above creates a new [Date](#) object and sets the variable [currentDate](#) to refer to it. The keyword **new** tells JavaScript to find the definition of the [Date](#) object and then construct one. We will look at objects in detail later in the text. Once we have our date object we can call **methods** on the object to make it do things for us.

```
function getTimeString() {  
  let currentDate = new Date();  
  let hours = currentDate.getHours();  
  let mins = currentDate.getMinutes();  
  let secs = currentDate.getSeconds();  
  let timeString = hours + ":" + mins + ":" + secs;  
  return timeString;  
}
```

The function [getTimeString](#) above creates a [Date](#) object and then uses the methods called

`getHours`, `getMinutes` and `getSeconds` to get those values out of it. The values are then assembled into a string which the function returns. We can use this function to get a time string for display.

Make Something Happen

Console Clock

Open beginntocloud.com and scroll down to the Samples section. Click **Ch01-What_is_the_cloud/Ch01-02_Console_Clock** to open the sample page. Then open the Developer Tools. Select the console tab.



Ch01_inset04_01 console clock page

The page shows an “empty” clock display. Let’s start by investigating the `Date` object. Type in the following:

```
> let currentDate = new Date();
```

Now press enter to create the new `Date` object.

```
> let currentDate = new Date();  
< undefined
```

This statement creates a new `Date` and sets the variable `currentDate` to refer to it. As we have seen before, the `let` statement doesn’t return a value, so the console will display “undefined”. Now we can call methods to extract values from the object. Type in the following statement:

```
> currentDate.getMinutes();
```

When you press Enter the `getMinutes` method will be called. It returns the minutes value of the current time and the console will display it.

```
> currentDate.getMinutes();  
< 17
```

The code above was run at seventeen minutes past the hour, so the value returned by `getMinutes` will be 17. Note that `currentDate` holds a “snapshot” of the time. To get an

updated date you will have to make a new [Date](#) object. You can also call methods to set values in the date too. The date contents will automatically update. You could use [setMinutes](#) to add 1000 to the minutes value and discover what the date would be 1000 minutes in the future.

The web page for the clock has the [getTimeString](#) function built in, so we can use this to get the current time as a string. Try this by entering a call of the function and pressing enter:

```
> getTimeString();  
< '13:18:17'
```

Above you can see a call of the function and the exact time returned by it. Now that we have our time we need a way of displaying it. The page contains a function called [showMessage](#) which displays a string of text. Let's test it out by displaying a string. Type the statement below and press enter

```
> showMessage("hello");
```



Ch01_inset04_02 test message

The web page now displays the string that was entered. Now we need a function that will display a clock tick. We can define this in the console window. Enter the following statement and press Enter:

```
> let tick = function(){showMessage(getTimeString());};  
< undefined
```

Take a careful look at the contents of the function and see if you can work out what they do. If you're not clear about this, remember that we want to get a time string and display it. The [getTimeString](#) function delivers a time string, and the [showMessage](#) function displays a string. If we have typed it in correctly, we should be able to display the time by calling the function [tick](#). Type the following statement and press Enter

```
> tick();
```



Ch01_inset04_03 time display

The time is displayed. The final thing we need for our ticking clock is a way of calling the tick function at regular intervals so that the clock keeps time. It turns out that JavaScript provides a function called `setInterval` that will do this for us. The first parameter to `setInterval` is a reference to the function to call. The second parameter is the interval between calls in thousands of a second. We want to call the function `tick` every second so type in the statement below and press Enter.

```
> setInterval(tick,1000);
```

This should start the clock ticking. The `setInterval` function returns a value as you can see below:

```
> setInterval(tick,1000);  
< 1
```

The return from `setInterval` is a value which identifies this timer. You can use multiple calls of `setInterval` to set up several timers if you wish. You can use the `clearInterval` function to stop a particular timer:

```
> clearInterval(1);
```

If you perform the statement above you will stop the clock ticking. You can make another call of `setInterval` to start the clock again. At the moment we have to enter commands into the console to make the clock. In the next chapter we'll discover how to run JavaScript programs when a page is loaded so that we can make the clock start automatically.

Arrow functions

If you look at lots of popular cartoon figures you will notice that some only have three fingers on each hand, not five. You might be wondering why this is. It is to reduce the “pencil miles” for the animators. The first cartoons were made from frames that were all hand-drawn by animators. They discovered that they could make their lives easier by reducing the number of fingers they had to draw. Fewer fingers meant fewer “pencil miles”.

The JavaScript arrow function is a way of reducing the “keyboard miles” of a developer. The character sequence `=>` provides a way to create a function without using the word `function` in the definition.

```
doAdditionArrow = (p1, p2) => {  
  let result = p1 + p2;  
  alert("Result: " + result);  
}
```

The JavaScript above creates a function called `doAdditionArrow` which is exactly the same as the original `doAdditon`. However, it is much quicker to type in. If the body of the arrow function only contains one statement you can leave off the braces that mark the start and end of the function body. And a single statement arrow function returns the value of the statement, so you can leave off the return keyword too. The code below creates a function called `doSum` which returns the sum of the two arguments.

```
doSum = (p1,p2) => p1 + p2;
```

We could call the `doSum` function as we would any other:

```
let result = doSum(5,6);
```

This would set the value of `result` to 11. You see the true power of the arrow notation when you start using it to create functions that are to be used as arguments to function calls.

```
setInterval(()=>showMessage(getTimeString()),1000);
```

This innocent looking statement repays careful study. It makes our clock tick. In the “Make Something Happen: Console clock” above we used the `setInterval` function to make the clock tick. The `setInterval` function accepts two arguments, a function to call and the interval between each function call. I’ve implemented the function to call as an arrow function containing a single statement which is a call to `showMessage`. The `showMessage` function has a single argument which is the message to be displayed. This is provided by a call to the `getTimeString` function.

Arrow functions can be confusing

JavaScript is not the first programming language that I've learned. It might not be your first language either. When I was learning JavaScript I found the arrow function one of the hardest things to understand. If you have seen C, C++, C#, or even Python programs before you will know about functions, arguments, parameters and return values already. This means that "traditional" JavaScript functions will be easy to grasp.

But you will not have seen arrow functions before because they are unique to JavaScript. And you can't easily work out what they do from seeing them in code. If you don't know what an arrow function does you will find quite tricky to understand what the creation of `doSum` above is doing. One way to deal with this would be to just regard arrow functions as a little extra provided by the language to make your life easier. If you don't mind doing the extra typing you can create all your programs without using the arrow notation. However, I think you should spend extra time learning how they work so that you can understand JavaScript written by other people.

There is one other aspect of arrow functions that can be confusing. There is a specific behavior that they have involving the `this` reference which it is important to know about. We will cover this in the section "The meaning of this" in Chapter 3.

When we started talking about JavaScript functions we noted that they are used to attach JavaScript code to events. The arrow function makes this very easy to do.

Make Something Happen

Arrow function Console Clock

You should already have the sample `Ch01-What_is_the_cloud/Ch01-02_Console_Clock` open. If not, open it. Then open the console tab and use the arrow function above to get the clock ticking. If the clock is already ticking you can reset everything by reloading the example page in the browser.

What you have learned

At the end of each chapter, we have a "what you have learned" section which sets out the major points covered in the text and poses some questions that you can use to reinforce your understanding.

- A web browser is an application that requests data from a web server in

the form of web pages. The connection between the two applications is provided by the internet.

- Web servers were originally single machines connected to the internet which were owned and operated by the owner of the site. The cloud made computing power into a resource that can be bought and sold. We can pay to have our web sites hosted in the internet. Page requests sent to the address of our site will be processed on machines at our service provider.
- The JavaScript programming language was invented to allow a browser to run program code that has been download from a website. It has since developed into a language that can be used to create web servers and free-standing applications.
- A JavaScript function is a block of code and a header which specifies the name of the function and any parameters that it accepts. Within the function the parameters are replaced by values which were supplied as arguments to the function call.
- When a running program calls a function the statements in the function are performed and then the running program continues from the statement after the function call. A function can return a value using the return keyword.
- When a JavaScript program runs inside the browser it uses an Application Programming Interface (API) to interact with the services the browser provides. The API is provided by many JavaScript functions.
- Operators in JavaScript statements perform an action according to the context established by the operands they are working on. As an example, adding two numbers together will result in an arithmetic addition, but adding two strings together will result in the strings being concatenated. If a numeric operation is attempted with operands that are not compatible the result will be set to the value “not a number”.
- Variables in JavaScript can hold values that indicate specific variable state. A variable that has not been assigned anything has the value “undefined”. A calculated value that is not a number (for example the result of adding a number to a string of text) will have the value NaN – short for Not A Number.
- Modern browsers provide a Developer Tools component that contains a console that can be used to execute JavaScript statements. The Developer Tools interface also lets you view the JavaScript being run inside the page and add breakpoints to stop the code. You can step through individual statements and view the contents of variables.

- A JavaScript function is represented by a function object. JavaScript manages these by reference so variables can contain references to functions. Function references can be assigned between variables, used as arguments to a function call and returned by functions.
- Function expressions allow a function to be created and assigned to a reference at any point in a program. A function expression used as an argument to a function call is called an *anonymous function* because it is not associated with any name.
- The JavaScript API provides a [Date](#) object which can be used to determine the current date and time and also allows date and time manipulation. The API also provides the functions [setInterval](#) and [clearInterval](#) which can be used to trigger functions at regular intervals.
- Functions can be defined using “arrow notation” which is shorter than the normal function definition. This is especially useful when creating functions to be used as arguments to function calls.

To reinforce your understanding of this chapter, you might want to consider the following "profound questions" about the cloud and what we do with it.

What is the difference between the internet and the web?

The internet is the technology that allows computers to communicate. The web is a service that uses the internet to link web browsers and web servers.

What is the difference between the cloud and the internet?

The internet is the networking technology that allows a program running on one computer to exchange data with a program running on another computer. We don't need the cloud to make an internet-based application. We just need two computers with internet connections. The cloud lets you replace a computer connected to the internet with a service that you have purchased from a cloud service provider. The cloud based server will have a network address which is used by the cloud service provider to locate the required service.

How does the cloud work?

A server hosted by a cloud service provider runs an operating system that allows it to switch between processes running services for different clients. At the front of the cloud service is a component which accepts requests and routes them to the process that provides the required service. The computing resources used by each process are monitored so that the services can be billed for the services they have provided.

What is the difference between a function and a method?

A function is declared outside of any objects. We have created lots of functions in this chapter. A method is a function which is part of an object. A [Date](#) object provides a [GetMinutes](#) method that returns the minutes value for a given date. This is called a method because it is part of the [Date](#) object. Methods themselves look like functions. They have parameters and can return values.

What is the difference between a function and a procedure?

A function returns a value. A procedure does not.

Can you store functions in arrays and objects?

Yes you can. A function is an object and is manipulated by reference. You can create arrays of references and objects can contain references.

What makes a function anonymous?

An anonymous function is one which is created in a context where it is not given a name. Let's take a look at the tick function we created for the clock:

```
let tick = function(){showMessage(getTimeString());};
```

You might think that this function is anonymous. However JavaScript can work out that the function is called "tick". If you look at the name property of the function you will find that it has been set to tick. But instead of creating a tick function we might use a function expression as an argument to the call of [setInterval](#):

```
setInterval(()=>showMessage(getTimeString()),1000);
```

The statement above feeds a function expression into [setInterval](#). The function expression does the same job as [tick](#), but now it is an anonymous function. There is no name attached to the function. Note that this statement uses the arrow notation to define the function.

Why do we make functions anonymous?

We don't have to use anonymous functions. We could create every function with a name and then use the name of that function. However, anonymous functions make life a lot easier. We can bind behaviors very tightly to the place they are needed. Also, if we are only ever going to perform a behavior once it is rather tedious to have to invent a function name for it.

Can an anonymous function accept parameters and return a result?

Yes it can.

Are arrow functions always anonymous?

An arrow function is simply a quick way of creating a function definition. Arrow functions can have names.

What happens if I forget to return a value from a function?

A function that returns a value should contain a return statement is followed by the value to be returned. However, if the function contains multiple return statements you might forget to return a value from one of them. The program will still run, but the value returned by the function at that point would be set to “undefined”.

What happens if I don't use the value returned by a function.

There is no need for a program to use the value returned by a function.

What does the let keyword do?

The let keyword creates a variable which is local to the block of code in which it is declared. When execution leaves the block the variable is discarded.

Do JavaScript programs crash?

This is an interesting question. With some languages the program text is carefully checked for consistency before it runs to make sure that it contains valid statements. With JavaScript, not so much. Using the wrong type of value in an expression, giving the wrong numbers of arguments to a function call or forgetting to return a value from function are all examples of program mistakes which JavaScript does not check for. In each of the above situations the program would not fail when it ran, instead the errors would cause variables to set to values like undefined or Not a Number. This means that you need to be careful to check the results of operations before using them in case a program error has made them invalid.

So the answer is that your JavaScript program probably won't crash, but it might display the wrong results.

2

Begin to Code: Chapter 2 Get Building Apps Into the Cloud and Games in

What you will learn

In the last chapter we discovered the origins of the cloud and ran some JavaScript code in a web page using the browser Developer Tools. We also learned a lot about JavaScript functions how to connect them to events

the Cloud

In this chapter we are going to take our JavaScript code and put it into the cloud for anyone to access. We are going start by getting the tools we are going to use and then move on to look at the format of the documents that underpin web pages. Then we are going to use JavaScript to add programmed behaviors to pages and discover how we can put our active pages into the cloud.

Rob Miles

Don't forget that you can use the Glossary to look up any terms that seem unfamiliar. Words defined in the glossary will be formatted like **this** in the text. Note that both **this** and **in** are in the glossary.

Working in the cloud

Before you start to build your applications, you need to find a nice place to work. There is a physical aspect to this. You work best if you are comfortable so a decent screen, nice keyboard and a responsive computer are all great things to have. However, there is also a “logical” element too. You need to find a place to store all the materials that you’re going to generate. You could just store all the files on your computer, but you might lose them if your machine fails. What’s more, if you make the wrong modifications to the only copy of a crucial file you can lose a lot of work very quickly (I have done this many times). So, let’s look at how we can manage our stuff. Starting with git.

Git

Git was created in 2005 by Linus Torvalds who was writing the Linux operating system at the time. He needed a tool that could track what he was doing and make it easy for him to share work with other people. So, he created his own. Git organizes data into “repositories”. A **repository** can be as simple as a folder containing a single file or it can be a **hierarchy** of nested folders containing thousands of documents. Git doesn’t care what the documents contain, or even what type of data they are. A repository can contain pictures, songs, and 3D designs along with software code.

You ask Git to “commit” changes you have made to the contents of a repository. When you do this the Git program searches through all the files in the repository and takes copies of the ones that have changed. These files, along with records of changes are then stored by Git in a special folder. The great thing about Git is that you can return to any of your commits at any time. You can also send people a copy of your repository (including the special folder). They can work on some of the files in the repository, commit their changes and send the repository back to you.

Git can identify which files they’ve changed and handle contentions. If both of you have changed a particular file Git can show the changes each person has made. These changes can be used to establish a definitive version of the file which is then stored back in the repository. If you think Git was a hard program to write, you are correct. It turned out to be tricky to manage file synchronization and change resolution. However, it makes it much easier for people to work together on shared projects.

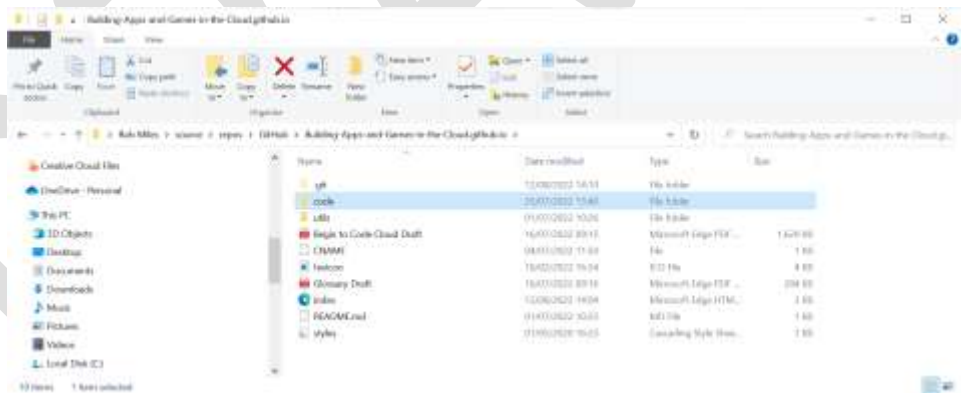


Figure 2.7 Ch02_Fig_01 Sample Repository

Figure 2.1 shows the contents of the repository containing all the sample code for this book. I have set the ‘**Hidden items**’ option in the **View** tab so that File Explorer shows hidden files. At the top the folder you can see a folder called **.git** which is the special folder created and managed by the git program. You can have a look inside if you wish, but you really shouldn’t change anything in it. When you copy a repository it is important that this folder be copied too.

The sample repository is exposed by GitHub as a web page, so it contains an index.html file. This is so you can view any of the sample code in your browser without downloading any samples onto your machine. Not all GitHub repositories are web pages, but it is very useful to be able to host a web site in this way. We will be doing this at the end of this chapter.

If we are going to work with repositories, we need to install the git software. This is a free download and there are versions for all types of computers.

Make Something Happen

Install Git

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

<https://git-scm.com>



Ch02_inset01_01 Git Install page

Now follow the installation process selecting all the default options.

Storing git repositories

You can use the git program on a single computer to manage your work. In this case you will put each repository in a folder somewhere. If you want to store your files in a central location, you can also set up a computer as a “git server”. This is a bit like a “web server for software developers”. You use the git program to send copies of your repositories over the network to the server and have them stored there. You can also “clone” a repository from the git server (this is called “checking out” a repository), work on it and then synch your changes with the original (this is called “checking in”). Git provides user management so individuals can have their own login names and be formed into teams that have access to particular repositories.

All the changes are tracked by git so that they can be reversed or examined in detail. This makes it much easier for developers to collaborate (which is what Linus Torvalds wanted at the start) as git can also detect multiple changes to the same file and insist that they are resolved to create a single definitive version.

GitHub and Open-Source Software

A git server makes it easy for programmers in a company to work together. But what if you want anyone in the world to be able to work on your project? Lots of important software used today, including operating systems, network tools and even games are now developed as **open-source** projects. All the software code that comprises these applications is stored openly, frequently in a git repository which is accessible to people who want to help with the project. Anyone can check out the repository, make some changes and then submit a “pull request” to the project owners. The project owners can “pull” in a copy of the changes. The changes could be a fix to a problem, a new feature, or even just improved error messages. If the changes check out these are then incorporated into the application which then gets that bit better.

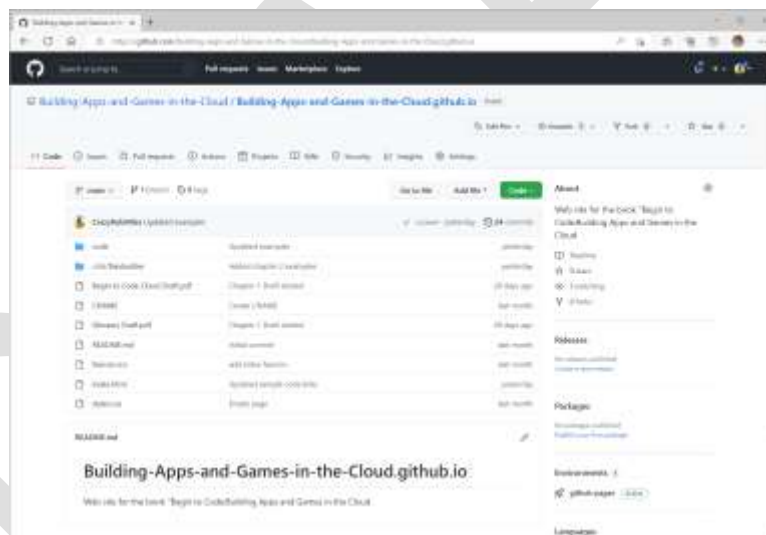


Figure 2.8 Ch02_Fig_02 Sample Repository on GitHub

Figure 2.2 above shows the sample repository as it appears on GitHub. GitHub also provides **organizations** as a way of managing projects. An organization can be created by a user and can contain multiple repositories. I’ve created one called “Building-Apps-and-Games-in-the-Cloud” for this book. The clock repository is one of several that are held in this organization. If you are storing repositories for a group or a large project which should not be directly associated with a particular GitHub user, you can create an organization to hold those repositories.

The clock repository is public, so anyone can look inside the files. They can even clone the

repository onto their computer, make some changes and send me a pull request. I have also added a license file which sets out what other people can do with the code, so you could regard the clock as a mini Open-Source Project. You can make private repositories which will not be visible to other GitHub users if you need them.

Programmer's Point

Open-Source projects are a great place to start your career

You might think that you must become a great programmer before you can start to make a name for yourself in software development. This is not true. You can add value to an open-source project well before you can create complete programs. You can learn a huge amount just by looking at code written by other people. And working out how the internal pieces of a system fit together is very satisfying, even if you don't know how the whole thing works.

An important part of being a successful developer is being able to work well with other people, so being part of an open-source project prepares you well for this. Many developers can also remember what it was like to start out and will be happy to help you improve, as long as you keep your input constructive and focused.

In addition, there are lots of things a project needs that have nothing to do with programming. A large project will need people to test things, write documentation, make artwork, create different language versions and so on. If you've got any of those skills, you could find yourself in high demand just for those. And that alone could open a totally different career path for you.

Many open-source projects (and lots of commercial ones) are hosted by GitHub. Companies rent space on GitHub rather than set up their own git server, but GitHub also offers comprehensive free services for open-source developers. It also offers a good service to individual developers; you can host your own private repositories on GitHub for your personal projects.

I strongly advise you to set up an account on GitHub and start using it for your projects. It will not cost you anything to do this. Whenever I start a new project one of my first thoughts is "what happens if I lose everything?". GitHub is a good answer to that question. If I put things into a GitHub repository they will be as safe as they can be. And if I make good use of the ability to commit changes at regular intervals I can save myself from losing work. Furthermore, once the repository is on GitHub I can invite other people into my project to work on it with them, or make it public so anyone can use it.

You don't need a GitHub account to make use of everything in this book, but if you want to get the most out of the contents you really should make one. It can completely change the way you work. If I want to do anything these days, from organize a party to write a book, I start by creating a GitHub repository for the project.

GitHub is a great place to store your data. However, it is also a great place to network with others. Each repository has a Wiki. This is a space for collaboratively creating documentation. A repository also has issue tracking discussions that people can use to report bugs and request features and you can create and manage projects within the repository.

GitHub also has a final ace up its sleeve. It can host web sites. You can take your web pages, put them into a GitHub repository and then make them available for anyone in the world to see. We will be doing this at the end of this chapter.

Make Something Happen

Join GitHub

If you don't want to join GitHub you can skip this section. You can still grab the sample repositories and work on the code but you won't be able to create your own repositories or use GitHub to host websites.

You will need an email address to create a GitHub account. Start by opening your browser and visit the web page: <https://github.com/join>



Ch02_inset02_01 GitHub join page

Enter your email address and click "Sign up for GitHub". It takes a few steps, and you have to wait for a message to validate your email address, but eventually you will end up at the dashboard for your GitHub account.

You are now logged in to the GitHub website and can work with your repositories and clone those of other GitHub users. If you want to use GitHub from the browser from another machine you will have to log in to the service on that machine.

You can work with your repositories using the web page interface. You can also enter git commands into your Windows terminal or MacOS console. However, we are going to use the Visual Studio Code program to edit and debug our programs and it can also talk to git and manage our repositories for us.

Programmer's Point

GitHub is also a social network.

You can regard GitHub as a social network. If you make something that you think would be useful to other people, you can share it by creating a public GitHub repository. Other GitHub members can download your repository and use it. They can also make changes and send you “pull requests” to signal that they have made a new version you might like.

Members can award repositories “stars” and comment on their contents. There are also lots of discussions going on. This means that GitHub is another place you can start to make a name for yourself. However, just as you should be cautious when you post details on other social network sites, you should also be aware of the potential for misuse that social networking frameworks provide.

Make sure not to give out too much personal information and ensure that anything you put into a public repository does not contain personal data. Don't put personal details, usernames, or passwords into program code that you put on GitHub. Later in the text we will discover how to remove personal information from projects that we want to make public.

Get Visual Studio Code

Now that we have git installed and working, we are going to install Visual Studio Code which we are going to use to create all our programs. It is free to download and versions are available for Windows, Macintosh and Linux based computers, including the Raspberry Pi. It also supports lots of extensions that can be used to extend its capabilities.

Make Something Happen

Install Visual Studio Code and clone a repository

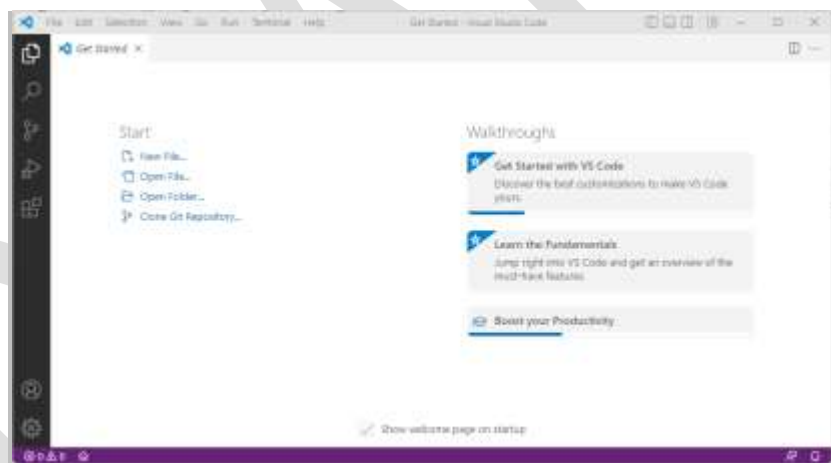
I'm going to give you instructions for Windows. The instructions for macOS are very similar. First you need to open your browser and visit the web page:

<https://code.visualstudio.com/Download>



Ch02_inset03_01 Visual Studio Install 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.



Ch02_inset03_02 Visual Studio install complete

Now that you have Visual Studio installed the next thing you need to do is fetch the sample files to work on. To do this click the Clone Git Repository link which is about halfway up the left-hand side of the page.

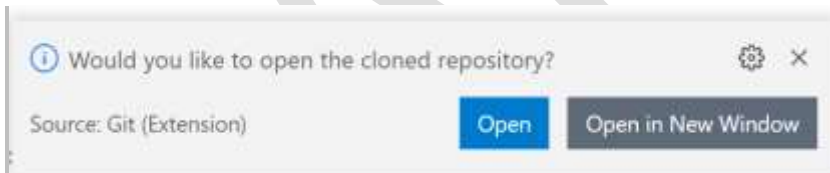


Ch02_inset03_03 Clone examples repository

<https://github.com/Building-Apps-and-Games-in-the-Cloud/Building-Apps-and-Games-in-the-Cloud.github.io>

Enter the address of the repository into the dialog box as shown above. Then click the “Clone from URL” button underneath the address. If you click the “Clone from GitHub” button underneath “Clone from URL” you will be prompted to log in to GitHub. This can be useful if you want send local repositories from your machine into GitHub, but you don’t need to do this to just fetch files.

The repository will be copied into a folder on your machine. The next thing that Visual Studio needs to know is the location of that folder. I have a special GitHub folder where I store my repositories. This is not in my OneDrive folder. Since I’m using GitHub to keep my files safe, I don’t need to use OneDrive to synchronize things.



Ch02_inset03_04 Clone complete

Once the files have been cloned Visual Studio Code offers you the chance to open the new repository. Click **Open** to do this. You will be asked to confirm that you trust the author. It is best to say yes here. Then Visual Studio will open the repository and show you the contents. On the far left are the tools you can use to work on the repository. Click the top one to select Explorer. The examples are in the code folder, organized by chapter. Click on **Ch02-Get_into_the_cloud** to open the folder and then open the “**Ch02-01_Simple HTML**” folder and then click the **index.html** file to open it in the editor.



Ch02_inset03_05 editing code

This is the first example file in this chapter. We will be looking at it later. Leave Visual Studio Code running, we will be using it again in a moment. The next thing to do is install our first Visual Studio Code extension.

Install the Live Server Extension

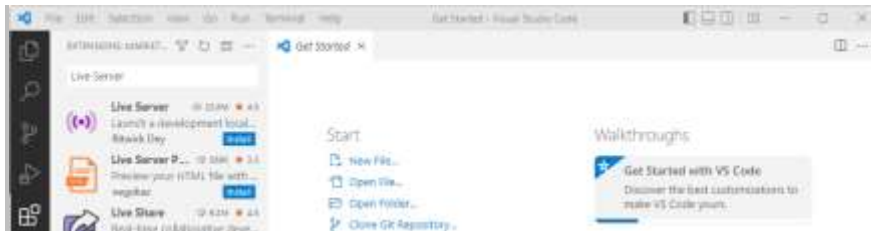
Before we can work with our web pages there is something we can do to make our job much easier. If you think about it, what we are going to do is edit a web page with Visual Studio Code, view the page in our web browser, edit it again and so on. We could do this by repeatedly saving the web page to a file and opening it by hand each time. However, this would be hard work and programmers hate hard work. What programmers do when faced with a problem like this is create a tool that will do the work for them.

Ritwick Dey is a programmer who solved this problem by creating the Live Server extension for Visual Studio Code. An extension is something you can add to Visual Studio Code to add a new feature. There are thousands of extensions available for download, we are going to use the Live Server extension. Then, when we want to view HTML in our browser we can just press a button to do this.

Make Something Happen

Install the Live Server extension

To begin, open Visual Studio Code. Then click the extensions button on the left-hand tool-strip (it is the fifth one down as highlighted below. The Extensions Marketplace opens. Type Live Server into the search box.



Ch02_inset04_01 Installing Live Server

The marketplace will show all the extensions with this name. Click the Install button on the one written by Ritwick Dey.



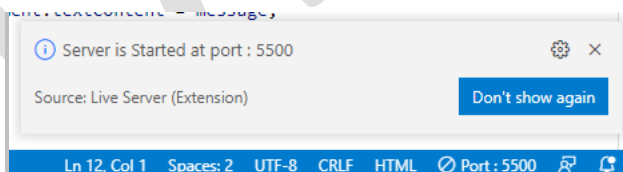
Ch02_inset04_02 Live Server installed

The extension will be installed and will start up each time Visual Studio Code is loaded. We can test it by using the clock page. If you have not left Visual Studio Code open from earlier, you should open the **"Ch02-01_Simple HTML"** folder in the **clock** repository and then click the **index.html** file to open it in the editor.



Ch02_inset04_03 VSC with live server ready

You can see the Go Live button on the bottom left of the screen in the blue border. Click the button and Go Live will now open the index file in your browser. You might see some messages from the firewall on your machine the first time you do this. You should allow Visual Studio Code access to the ports that it needs.



Ch02_inset04_05 VSC live server starting

You will also see a message from Live Studio telling you the server has started and specifying the network port that it is using. Then the browser opens and displays the page.



Ch02_inset04_06 VSC live server web page

Above you can see the page as displayed by the browser. Note that the address of the page starts with 127.0.0.1 which indicates that the page is being hosted by your computer. If you make changes to index.html and then save them the browser will automatically reload the updated page. This is a very useful extension. It has been downloaded more than 23 million times. It turns out that writing extensions for Visual Studio Code is also a great way to make a name for yourself. Next, we need to build an understanding of the contents of the web page.

How a web page works

We have been using web pages in our browsers for years. In chapter 1 we saw that a web page can contain JavaScript code. Now it is time to dig deeper and consider what makes a web page. We are not going to go into too much detail, there are books specifically about these topics that are much thicker than this one. However, you may think you already know what a web page is, but I'd be most grateful if you would read this section anyway. You may find a few things you didn't know.

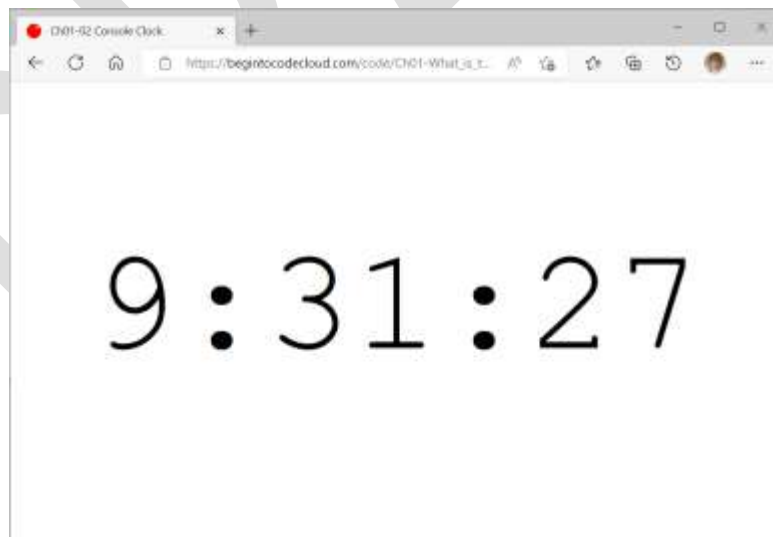


Figure 2.9 Ch02_Fig_03 Ticking Clock

A web page is a “logical document”. What do I mean by that? Well, a book is a **physical** document. It exists in the real world. We can read it, add annotations, leave it on the bus and do everything else that you can do with a physical book in the real world. An electronic book (or e-book) is a **virtual** document. It is something created to play the part of a physical book in a virtual environment created by a computer. We can read it, annotate it, and if we delete the file containing the book, we could even manage to lose it. A web page is a **logical** document. It is a thing created by software and hosted on a computer. A web page has no physical counterpart. There is no physical version of a book that contains a ticking clock, as you can see in Figure 2.3 the best we can do is a stationary image.

Loading a page and displaying it

The starting point for a web page is a file of text which is hosted on a server. When you visit a website, for example www.robmiles.com, the browser looks for a file called `index.html` at that location which contains the start page of the site. The text in this file describes the logical document that makes up the web page. The browser reads the file of text from the server and uses it to build the logical document that is the web page. At this point you might say “Aha! The file of text on the server describes the web page so a web page is just a file of text.” Well, no. A web page could contain JavaScript code that runs when the document is loaded into the browser. That code can change the contents of the logical document and even add new things. The logical document that the HTML describes is held by the browser in a structure called the **Document Object Model** (DOM). The DOM is a very important part of web programming. Our JavaScript programs will interact with the DOM to display output.

Once the browser has built the DOM it draws it on the display. The browser program then repeatedly checks the DOM for changes and redraws it if any changes are found. This is how the ticking clock we created in chapter 1 works. There is a JavaScript function running which changes something in the document and the browser redraws the page to reflect the changes. We don’t have to do anything to trigger a redraw, our code can change the contents of the logical document and the updated version of the page is displayed automatically.

A logical document can contain images, sounds and videos which are all updated automatically. The initial contents of the document objects are expressed using a language called Hypertext Markup Language or HTML. Let’s look at that next.

Hypertext Markup Language (HTML)

We can discover what Hypertext Markup Language really is by unpicking its name. Let’s start with Hypertext. Remember that we can call things **logical** to indicate that they have no counterpart in real life. **Hypertext** is a logical version of text. Normal text, whether it is printed or displayed on a screen, is something you read from beginning to end. Hypertext can only be displayed by a computer. This is because hypertext can contain **hyperlinks** which refer to other documents. You can start reading one document, open a hyperlink and be moved to a completely different one, perhaps served by a completely different computer in a different country. When hypertext and

hyperlinks were invented, it was thought cool to put the word hyper in front of them to make them sound impressive. So that is where the name came from. At least, that's what I think.

So, the word hyperlink in HTML means that the aim of the language is to express a page that can contain hyperlinks. HTML was invented to make it easier to navigate reports. Before hyperlinks, if a report contained a reference to another report you would have to go and find that report and open it to read it. After hyperlinks you could just follow a link in the original report. Hypertext was designed to be extensible, so that it would be easy to add new features. The features provided by modern web pages are way beyond any foreseen by Tim Berners-Lee, the inventor of HTML, but the fundamental content of a page remains the same.

The word **markup** refers to the way that an HTML document separates the intent of the author from the content in the page. Let's look at a tiny web page to discover how this works.

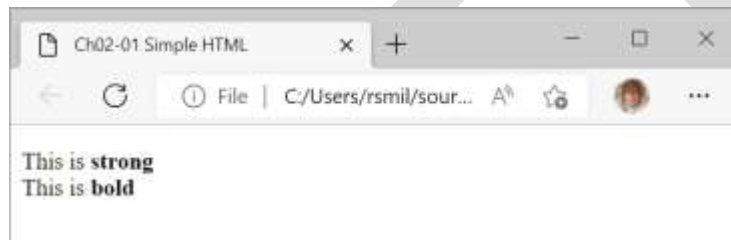


Figure 2.10 Ch02_Fig_04 Simple HTML

Figure 2.4 shows a tiny web page. It only contains six words. Let's look at the HTML file behind it and discover the role of markup in creating the page contents:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Ch02-01 Simple HTML</title>
</head>
<body>
  <p>
    This is <strong>strong</strong><br>
    This is <b>bold</b><br>
  </p>
</body>
</html>
```

This is the html file that describes the page in Figure 2.4. The most important characters in the file are the < and > characters that mark the start and end of HTML element names. The < and >

are called **delimiters**. They *define the limits* of something. An **element** is a thing in the document that the browser knows how to work with. Elements can have attribute values. These are name-value pairs included in the definition of the element. The `<html lang="en">` element above has an attribute called `lang` which gives the language of the html page. The language code `en` means English.

Elements can be containers. A container starts with the name of the element and ends with the name preceded by a forward slash (/). You can see that the `<title>` element contains the text that is to be used as the title of the web page. You can see this title on the top of the web page in Figure 2.4. The title information is part of the header for the page, which is why it is enclosed in the `<head>` element.

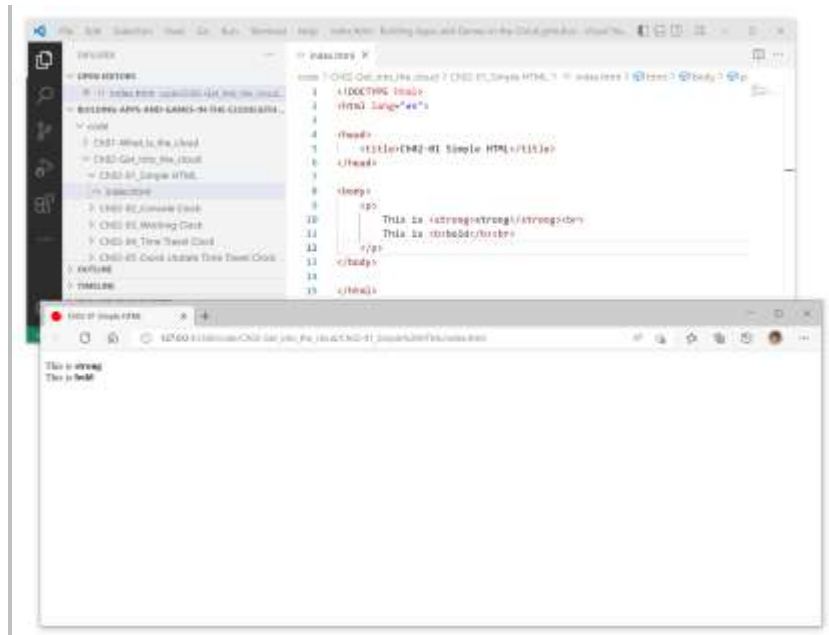
The `<body>` element contains all the elements to be drawn by the browser. The `<p>` element (short for paragraph) groups text into a paragraph. The `<bold>` and `` elements give formatting information to the browser. However, an element can exist as the starting element name, without another to mark the end of the element. The `
` element tells the browser to add a line break in the text. You don't need to add a `</br>` element to a page to mark the end of a line break.

Now we can look at the last word in the phrase "Hypertext Markup Language". The word language means that we are going to use HTML to express things. HTML is very specific (it is being used to tell a browser how to build a logical document) but it is a language.

Make Something Happen

Web page editing

If you have been following this exercise you will already have the first sample file open in your browser. If not, use Visual Studio Code to open the "**Ch02-01_Simple HTML**" in the examples and then click the **index.html** file to open it in the editor. Then click **Go Live** to open the page in the browser.



Ch02_inset05_01 Visual Studio and Browser

Your desktop should now have both Visual Studio and your browser on it, as shown above. Now go back to Visual Studio Code and add the following element into the body element of the page:

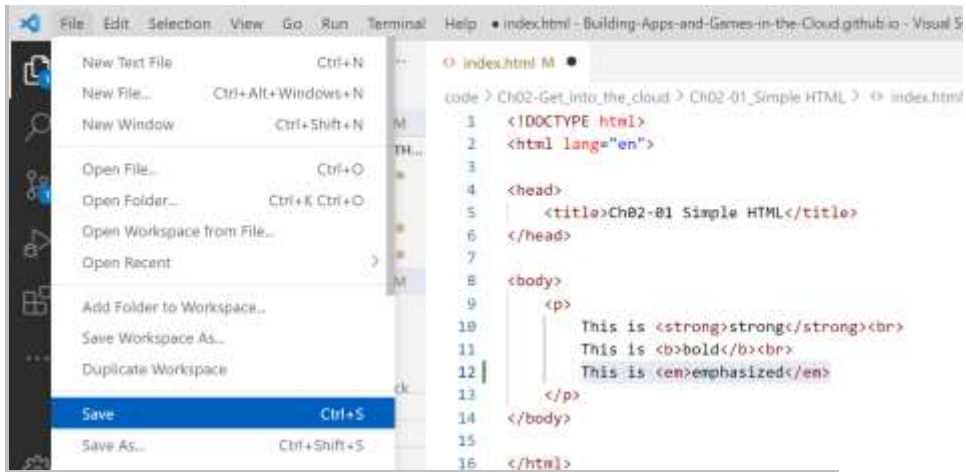
This is `emphasized`

Visual Studio Code should now look like this:



Ch02_inset05_02 web page edits

Now select **File>Save** from the Visual Studio Code menu bar to save the updated file



Ch02_inset05_03 edit save

Because you are using Go Live you should see the page in the browser update automatically and display the new text on the page.



Ch02_inset05_04 updated page

This is a very nice way to work on web pages. Each time you save your file the web page updates automatically.

CODE ANALYSIS

Investigating HTML

If this is your first brush with HTML you may have some questions.

Question: What is the difference between bold and strong?

The HTML file marks one piece of text as bold and another as strong. But in Figure 2.2 they both look the same. You might think that this means that bold and strong are the same thing. However, they are intended to be used for different kinds of text. Text that needs to stand out should be formatted bold. Text that is more important than the text around it should be formatted strong. I would print my name in bold (because I would like it to stand

out). I would print “Do not put your head out of the window when the train is moving” in strong because that is probably more important than the text around it.

Remember that HTML just contains instructions to the browser to display things in certain ways. The browser has a default behavior (to display bold and strong as bold text) but you can change this for your web pages by adding styles, of which more later.

Question: How do I enter a < or a > into the text?

Good question. 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. Some useful ones are:

```
&lt; &gt; &amp;
```

You can find a list of all the symbols here: <https://html.spec.whatwg.org/multipage/named-characters.html> You can also use symbols to add emoticons to your pages. Take a look here for the codes to use <https://emojiguide.org/>

Question: What happens if I mis-spell the name of an element?

If the browser sees an element it doesn't know it will just ignore it.

Question: What happens if I get the nesting of the elements wrong?

If you look at the sample code, you will see that some elements are inside others. The <p> element is inside the <body> for example. If you get the nesting wrong (for example put the </body> element inside the <p> the browser will not complain and the page will display, but it might not look how you were expecting.

Question: What does the <!DOCTYPE html> element mean?

Good question. The very first line of a resource loaded from a web server should describe what it contains. The resource could contain an image, a sound file, or any number of other kinds of data. The !DOCTYPE element is used to deliver this information. The browser doesn't always use this. A browser will try to display any file of text it is given, but it is very useful to add the information.

Question: How do I put JavaScript into a web page?

You use the <script> element to embed JavaScript into a page.

Question: Are there other kinds of markup language?

Yes there are. There is XML (eXtensible Markup Language) which is used for expressing structures of data. There are also lots of others which have been developed for particular applications.

Question: How to I stop the Go Live server?

You might want to stop the Go Live server and open a web page from a different index file. You can do this by restarting Visual Studio Code, but you can also press the close button next to the port number on the bottom right of the Visual Studio Code window.

Make an active web page

We have reached a pretty powerful position. We have tools that we can use to create and store web resources and we are building an understanding of the way that web pages are structured. Now we are going to take another big step forwards. We are going to discover how a JavaScript program can modify the contents of the document object and change the appearance of the web page. This is how JavaScript programs running in the browser communicate with the user.

Interact with the document object

Make Something Happen

Interact with a web page

If you have been following this exercise you will already have the console clock open in your browser. If Go Live is displaying the page, stop it from running by pressing the close button next to the port number at the bottom of the Visual Studio Code window. If not, use Visual Studio Code to open the “**Ch02-02_Console Clock**” folder in the examples and then click the **index.html** file to open it in the editor. Then click **Go Live** to open the page in the browser. Now open the Developer Tools and then select the **Elements** tab. This shows us the elements in the document:



Ch02_inset06_01 Clock page elements

In the figure above I've resized the Developer Tools part of the screen to give me more space. On the left of the window, you can see the page as displayed by the browser. On the right you can see a display of the contents of the web page. One item has been selected in

the view. This is a paragraph which has an attribute called `id` which is set to the value `timePar`. This page looks a lot like the source code of the page, but it is actually a view of the elements in the **Document Object Model** (DOM) that was created from the html file.

```
<p id="timePar" class="clock">0:0:0</p>
```

Above you can see the element that is highlighted in the image. The clock program changes the content of this element to display the time. The program looks for the element with an `id` attribute of `timepar` and then displays the time in this element. In the last chapter we used the `showMessage` function to display a message on the page. Lets look at how it works.

```
showMessage("console hello");
```

This is how we call `showMessage`. We give it an argument which is the message to be shown on the screen.



Ch02_inset06_02 Showing console hello

You can see the effect of the call of `showMessage` above. The message is displayed because the browser redrew the document and the data in the document object has changed. Now let's take a look at the `showMessage` function itself.

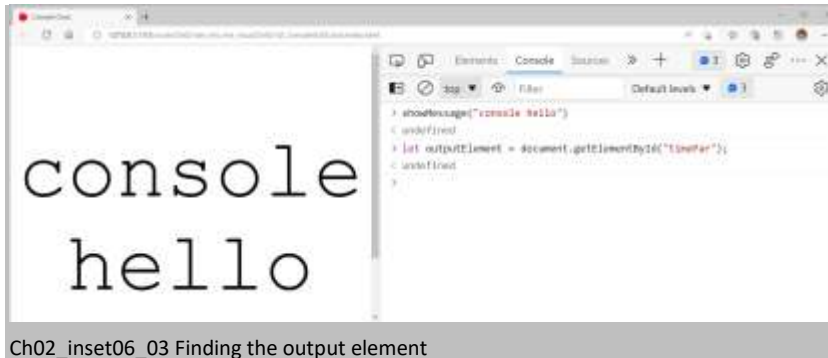
```
function showMessage(message) {  
  let outputElement = document.getElementById("timePar");  
  outputElement.textContent = message;  
}
```

You can see the function above. It contains just two statements. The first statement finds an HTML element in the document and the second statement sets the `textContent` property of this element to the message that was supplied as a parameter. That sounds simple enough, especially if we say it very quickly. Or not. Let's break it down into a series of steps and enter them into the console. Select the Console tab in the Developer Tools window and enter the following statement:

```
let outputElement = document.getElementById("timePar");
```

This statement finds the element in the document that displays the output (that's what

`getElementById` does). The `getElementById` method is provided by the DOM to search for elements by name. The element we have asked it to look for is a paragraph that has an `id` property set to `timePar`. The value returned by `getElementById` is assigned to a variable called `outputElement`. Press enter to perform the statement.

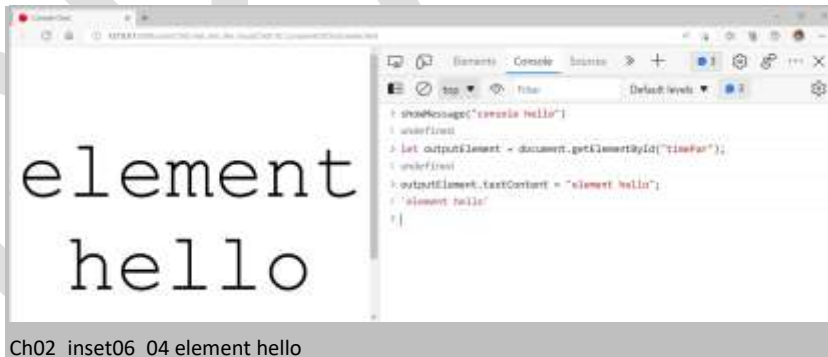


Ch02_inset06_03 Finding the output element

We know that `let` does not return a value, so the console displays "undefined". The display has not changed because all we have done is obtain a reference to a paragraph in the document. We need to change the text in that paragraph. Enter the following statement:

```
outputElement.textContent = "element hello";
```

This statement uses the `outputElement` reference that we have just created. It puts the string "hello" into the `textContent` property of the element that `outputElement` refers to. Press enter to see what that does.



Ch02_inset06_04 element hello

The page now shows the message "element hello" because the `textContent` property of the `outputElement` is now "element hello". The `showMessage` function performs these two steps and sets the `textContent` property to a parameter it has been supplied with, so we can use `showMessage` to show any message we like.

CODE ANALYSIS

Documents objects and JavaScript

The way that JavaScript programs can interact with document is very powerful. But it can also be very confusing. You might have some questions.

Question: Why is the text that we are displaying so large?

This is a good question. All the other text on the web pages we have made has been tiny. But the time message in our clock is huge, and in a different font. How does this work? It works because we are using a feature called a **stylesheet**. Let's take a look at the HTML that defines the paragraph containing the clock output:

```
<p id="timePar" class="clock">0:0:0</p>
```

This is a paragraph that initially contains "0:0:0". It has the id `timePar` (this is how our JavaScript finds the paragraph). It also contains another attribute. It has a `class` attribute which is set to the value of `clock`. The class attribute tells the browser how to display an item when it is drawn. Alongside the HTML file containing the clock web page is a field which defines the styles for this page. There is a link element in the `head` element for the clock web page that tells the browser which stylesheet file to use.

```
<link rel="stylesheet" href="styles.css">
```

This document is using the `styles.css` file. Inside this file you can find the following style definition:

```
.clock {  
  font-size: 10em;  
  font-family: 'Courier New', Courier, monospace;  
  text-align: center;  
}
```

This style information defines the clock **class**, which can then be assigned to elements in the web page. It sets the size, the font, and the alignment. When a web page is loaded the browser also fetches the stylesheet file that goes with it. You can use stylesheet files to separate the formatting of a page from the code that creates it. If the designer and the programmer agree on the names the classes to use the designer can create the styles and the programmer can create the software.

It is possible for a running program to change the style class of an element. You see this happen when invalid items on a web form are highlighted in red. What has happened is that the JavaScript processing the form has changed the style class of an element to one which has a color setting of red.

Question: What if we try to find a document element that is not there?

```
let outputElement = document.getElementById("timeParx");
```

The statement above is valid JavaScript and is intended to get a reference to the element with the id `timePar`. However, it will not do what we want because we have mis-typed the id and there is no element in the web page with the id `timeParx`. When the statement runs the `getElementById` function returns the value of `null`. This means that `outputElement` is set to `null`. The value `null` is another JavaScript special value. We've seen ones called `nan` (not a number) and `undefined` (I have not been given a value). The value of `null` is how `getElementById` can say "I looked for it, but I couldn't find it – therefore I'm giving you a reference back that explicitly means that it was not found". If a program tries to set a property on a `null` reference we get an error that stops the program:

```
> let outputElement = document.getElementById("timeParx");  
< undefined  
> outputElement  
< null  
> outputElement.textContent = "hello";  
✖ ▶ Uncaught TypeError: Cannot set properties of null (setting 'textContent')  
    at <anonymous>:1:27  
>
```

Ch02_inset07_01 null reference error

Above you can see the effect of trying to use a `null` reference. The message in red means that the program would stop at this point. In other words, any statements after the one that tries to set the `textContent` property on a null reference would not be performed. Later we will discover how we can detect null references and deal with statements that fail like this.

Question: What if we try to use an element property that is not there?

```
outputElement.textContentx = "hello";
```

The statement above is also legal JavaScript. But it won't display the message hello. This is because it sets a property called `textContentx` rather than the correct `textContent` one. What happens next is really very interesting. You don't get hello displayed because you've written to the wrong property. Instead, JavaScript creates a new property on the `outputElement` object which is called `textContentx`. It then sets the content of this property to "hello". This is a powerful feature of the JavaScript language. It means that code running in the web page can attach its own data to elements on the page. We will explore this feature later in the book.

Leave this page open because we are going to be working on it in the next "Make Something Happen".

Web pages and events

We are going to make a web page that contains active content which runs when the page is

loaded. The previous pages have contained JavaScript functions, but we have had to run them from the Developer Tools console. Now we are going to give them control to make truly interactive web sites. To do this we are going to bind a function to the event that is fired when a web page is loaded by the browser. In chapter 1 in the section “JavaScript Heroes: Functions” we saw how we can pass an event generator a function reference so that the function is called when the event occurs. We used it to make a ticking clock. The clock contained a function `tick` which could get the time and display it.

```
function tick(){
    let timeString = getTimeString();1
    showMessage(timeString);2
}
```

You can see the `tick` function above. I’ve expanded it slightly from the version in chapter 1 to make it clear how it works. The first statement in the function gets the time into a string called `timeString` and the second statement shows it on the screen. Each time `tick` is called it will show the latest time value. To make the clock display update continuously we need a way of calling the tick function at regular intervals. We can use the `setInterval` function to do this:

```
setInterval(tick,1000);
```

The `setInterval` function is called with two arguments. The first argument is a reference to `tick`, and the second argument is the interval between calls, in this case 1000 milliseconds. This will make our clock tick every second. The `setInterval` function causes an event to be created every second, so the clock will update every second. What we need next is a way of calling this function to start the clock each time the clock web page is loaded.

```
function startClock(){
    setInterval(tick,1000);
}
```

¹ *Get the time*

² *Display the time*

Above you can see a function called [startClock](#). If we can find a way of calling this function when the clock web page is loaded we can make a clock that starts when the page is loaded. When we began learning JavaScript in chapter 1 we saw the importance of learning the Application Programming Interface (API) that provides functions you can perform. Now we are starting to see the importance of another kind of interface; the one provided by properties of elements in the web page itself. We've seen how we can add properties to elements in an HTML document. Each element supports a set of properties. Some of the properties can be bound to snippets of JavaScript.

```
<body onload="startClock();">
```

The body element above now has an attribute called [onload](#) which is the string "startClock();". Properties with names that start with the word "on" are event properties that will be triggered when the event occurs. The [onload](#) event is triggered when the body of a page is loaded. When the page is loaded the string of JavaScript is performed by the browser in just the same way as the browser performs commands that we have typed into the console.

Make Something Happen

Make a ticking clock

If you have been following this exercise you will already have the console clock open in your browser. If not, use Visual Studio Code to open the "**Eg 01 Console Clock**" folder in the **clock** repository and then click the **index.html** file to open it in the editor. You are going to add some functions to the JavaScript code in the web page, so scroll to that part of the document.

```
function tick(){
    let timeString = getTimeString();
    showMessage(timeString);
}

function startClock(){
    setInterval(tick,1000);
}
```

These are the two functions that make the clock work. Type them into the [index.html](#) page inside the `<script></script>` part of the page, near the two existing functions.

Now we need to modify the [body](#) element to add the [onload](#) attribute that will call the [startClock](#) function. Navigate to line 10 in the file and modify the statement as follows:

```
<body onload="startClock();">
```

Now the body element has an `onload` attribute which will call the `startClock()` function when the page is loaded. Now press the Go Live button to open the page in the browser. You should see the clock start ticking.

CODE ANALYSIS

Events and web pages

Events are great fun, but you might have some questions about them:

Question: My clock is not ticking. Why is this?

There are a number of reasons why your clock might not tick. If you mis-spell the name of any of the functions they might not be called correctly. For example, if you call the starting function `StartClock` (with an upper-case S at the beginning) then this will not work, because the onload event is expecting to call a function called `startClock`. If you get completely stuck, head over to the **Eg 02 Working clock** folder where there is a working version of the clock.

Question: What is the difference between an attribute and a property?

Good question. A property is associated with a software object. We have seen how objects can have properties that we can access when our programs run. For example, we looked at the name property of a function object in chapter 1 in the section Make Something Happen - Fun with Function Objects. An attribute is associated with an element in a web page. The `body` element can have an `onload` attribute.

Question: Can you run more than one function when a page loads?

You can do this, but you wouldn't do it by adding multiple onload attributes. Instead you would write a single function that runs all the functions in turn, and connect that to the onload event.

Making a time machine clock

Now that we know how to make a clock, we are going to make a clock that lets us travel through time. Sort of. In chapter 1 in the section Make Something Happen - Console Clock we noticed that the `Date` object provides methods that could be used to set values in a date as well as read them back. If we use this to add 1000 minutes to the minute value, the `Date` object will work out the date and time 1000 minutes into the future.


```
let d = new Date();3  
let mins = d.getMinutes();4  
let mins = mins + 1000;5  
d.setMinutes(mins);6
```

The code above shows how this would work. The first statement creates a variable called `d` that refers to an object containing the current date. The second statement creates a variable called `mins` that holds the number of minutes in the date stored in `d`. The third statement adds 1000 to the value of `mins`. The fourth statement sets the minutes value of `d` to the value in `mins`. This moves the date in `d` 1000 minutes into the future. The `Date` object sorts out the date value, updating the hours and even the day, month and year if required.

We can use this to make a time travel clock which is fast or slow by an amount that we can nominate. At some times of the day, perhaps the morning, we can make the clock go fast so we are not late for anything. At other times, perhaps when it is time to go to bed, we can make the clock go slow, so we can go to bed a little later.



Figure 2.11 Ch02_Fig_05 Time Travel Clock

Figure 2.5 above shows how it would be used. The user clicks the buttons to select a fast clock, a

³ Make `d` refer to a new `Date` object

⁴ Extract the minutes from the date

⁵ Add 1000 to the minutes value

⁶ Set the minutes in the future

slow clock, or a normal clock.

Add buttons to a page

The first thing we need to do is add some buttons to the page for the user to press. We do this with the `button` element.

```
<p>
  <button onClick="selectFastClock();">Fast Clock</button>
</p>
```

Above you can see a paragraph that contains a `button`. The button encloses the text that will appear on the button on the page. The button element can have an `onclick` attribute which contains a string of JavaScript to be performed when the button is clicked. When this button is clicked a function called `selectFastClock` is called. Now we need to create some code to put inside this function that will make the clock five minutes fast when it runs. We can do this by creating a **global** variable.

Share values with global variables

Up until now every variable we have created has been used inside a function body. We have used `let` to create the variable. A variable declared with `let` ceases to exist when the program execution exits the block in which the variable was declared. So when the function completes the variable is discarded. Most of the time this is just what you want. It is best if variables don't "hang around" after you have finished with them. I'm very partial to using a variable with the identifier `i` for counting. This is because I am a very old programmer. However, I don't want an `i` used in one part of the program to be confused with one used somewhere else. I like the idea of a variable disappearing as soon as the program leaves the block where it was declared.

However, in the case of the minutes offset value we want to share the value between functions. The `minutesOffset` mustn't disappear when the program exits a function where it is used. We can't declare `minutesOffset` in just one function, we must declare it so that it can be shared between all functions. We must make it **global**.

```
var minutesOffset = 0;
```

The variable `minutesOffset` above is not declared inside any code block. It is declared outside all

the functions. It is also declared using `var` rather than `let`. This means that the variable can be used in any of the functions that follow it. The value in the variable will be shared by all the functions, which is exactly what we want. If we change the value in `minutesOffset` the next time the clock is updated by the tick `function` it will draw the new time.

Programmer's Point

Global variables are a necessary evil

When you make a variable global by declaring it outside any function, you lose control of it. What do I mean by this? Well, suppose I'm working with a bunch of programmers, each of them writing some of the functions in my JavaScript application. If I declare all the variables in my functions by using `let` I can be sure that those variables can only be changed by me. I can also be sure that I won't change any values in other functions.

However, if a variable is made global it is possible for code in any of the functions view it and change it, which might lead to mistakes and makes the program less secure.

Some things must be global. It would be very hard to make the clock work without creating a global variable called `minutesOffset`. But when you write code you should start by making the variables as local as possible (by declaring them using `let`) and then make things global if you have a need for this.

JavaScript give you control of variable visibility. We will discover how to do this in the next chapter in the section "JavaScript Heroes: `let`, `var` and `const`".

```
function selectFastClock() {  
  minutesOffset = 5;7  
}
```

Above you can see the code for the `selectFastClock` function. When the function runs it sets the variable `minutesOffset` to 5. The contents of the `minutesOffset` variable are added to the minutes value when the time is displayed.

⁷ Set the minutes offset to 5

```

function getTimeString() {
    let currentDate = new Date();8
    let displayMins = currentDate.getMinutes()
                        + minutesOffset;9
    currentDate.setMinutes(displayMins);10
    let hours = currentDate.getHours();
    let mins = currentDate.getMinutes();
    let secs = currentDate.getSeconds();

    let timeString = hours + ":" + mins + ":" + secs;11
    return timeString;12
}

```

Above is a modified version of the `getTimeString` function. This adds the value of `minutesOffset` onto the minutes in the time string. This means that when the fast button is clicked the clock will display the time five minutes into the future. The page also has button handlers for `selectSlowClock` and `selectNormalClock` which set the value of `minutesOffset` to the appropriate values. The complete HTML file for the Time Travel Clock is shown below. You can view the code running in the example **Eg 03 Time Travel Clock**.

```

<!DOCTYPE html>
<html>

<head>
    <title>Time Travel Clock</title>
    <link rel="shortcut icon" type="image/x-icon" href="favicon.ico">
    <link rel="stylesheet" href="styles.css">
</head>

<body onload="startClock();">
    <p id="timePar" class="clock">0:0:0</p>

    <p>

```

⁸ *Get the date*

⁹ *Calculate the new minutes*

¹⁰ *Set the new minutes value*

¹¹ *Build the time string*

¹² *Return the time string*

```
<button onclick="selectFastClock();">Fast Clock</button>
</p>
<p>
  <button onclick="selectSlowClock();">Slow Clock</button>
</p>
<p>
  <button onclick="selectNormalClock();">Normal Clock</button>
</p>

<script type="text/javascript">

  var minutesOffset = 0;

  function selectFastClock() {
    minutesOffset = 5;
  }

  function selectSlowClock() {
    minutesOffset = -5;
  }

  function selectNormalClock() {
    minutesOffset = 0;
  }

  function tick() {
    let timeString = getTimeString();
    showMessage(timeString);
  }

  function startClock() {
    setInterval(tick, 1000);
  }

  function getTimeString() {
    let currentDate = new Date();
    let displayMins = currentDate.getMinutes() + minutesOffset;
    currentDate.setMinutes(displayMins);
    let hours = currentDate.getHours();
    let mins = currentDate.getMinutes();
    let secs = currentDate.getSeconds();
    let timeString = hours + ":" + mins + ":" + secs;
    return timeString;
  }
}
```

```
function showMessage(message) {  
  let outputElement = document.getElementById("timePar");  
  outputElement.textContent = message;  
}  
</script>  
</body>  
</html>
```

CODE ANALYSIS

Time Travel Clock

You may have some questions about the time travel clock:

Question: Can we make the display update immediately after a button is pressed?

There is a problem with the time travel clock. It takes a while to “catch up” when you click a button. You have to wait up to a second before you see the time change to reflect a new offset value. We can fix this by making the [selectFastClock](#), [selectSlowClock](#) and [selectNormalClock](#) functions call the [Tick](#) function once they have updated the offset.

```
function selectFastClock() {  
  minutesOffset = 5;  
  tick();  
}
```

Now, when the user clicks the button the clock updates instantly. You can find this version in the example **Eg 04 Quick Update Time Travel Clock**

Question: Can we make the clock display change color to indicate whether the clock is fast or slow?

Yes we can. We could have a different style class of the display paragraph for each of the different clock options.

```
.normalClock, .fastClock, .slowClock {  
  font-size: 10em;  
  font-family: 'Courier New', Courier, monospace;  
  text-align: center;  
}  
  
.normalClock{
```

```

    color: black;
}

.fastClock {
    color: red;
}

.slowClock {
    color: green;
}

```

Above you can see a stylesheet that creates three styles, [normalClock](#), [fastClock](#) and [slowClock](#). You can see which settings are shared by all the styles and which just set the specific colors. The fast clock is displayed in red and the slow one in green. We can now set the style class to the appropriate style when the selection button is pressed.

```

function selectFastClock() {
    let outputElement = document.getElementById("timePar");
    outputElement.className = "fastClock";
    minutesOffset = 5;
    tick();
}

```

An HTML element has a [className](#) property which is set to the name of the class. We can change the style class by changing this name. The [selectFastClock](#) function sets the [className](#) for the [outputElement](#) to "fastClock" so that the [fastClock](#) style class is used to display it. So the text now turns red when the clock is fast.

Question: Is there a way to write this program without using a global variable?

At the moment the time travel clock uses a global variable called [minutesOffset](#) to determine whether the clock is fast or slow. Global variables are something to be avoided if possible, but how can we do this? It turns out that we can.

We've just made a modification that changes the [className](#) property of the time paragraph to select a different display style depending on whether the clock is fast, slow or normal. We can use the value of the [className](#) property on the time paragraph to set the time offset as well.

```

function getMinutesOffset(){
    let minutesOffset = 0;
    let outputElement = document.getElementById("timePar");
    switch(outputElement.className) {
        case "normalClock": minutesOffset = 0;
        break;
        case "fastClock": minutesOffset = 5;
        break;
        case "slowClock": minutesOffset = -5;
        break;
    }
}

```

```
    return minutesOffset;
}
```

The function `getMinutesOffset` above uses the JavaScript **switch** construction to return an offset value which is 0 if the `className` property of the `timePar` element is `normalClock`, 5 if the `className` is `fastClock` and -5 if the `className` is `slowClock`. This can be used in `getTimeString` to calculate the time to be displayed.

```
function getTimeString() {
    let currentDate = new Date();
    let minutesOffset = getMinutesOffset();
    let displayMins = currentDate.getMinutes() + minutesOffset;
    currentDate.setMinutes(displayMins);
    let hours = currentDate.getHours();
    let mins = currentDate.getMinutes();
    let secs = currentDate.getSeconds();
    let timeString = hours + ":" + mins + ":" + secs;
    return timeString;
}
```

This version gets the value of the minutes offset and then uses it to create a time string with the required offset. There is now no need for a global `minutesOffset` variable.

This is a very good way of solving the problem. The setting is held directly on the element that will be affected by it. There is also no chance that the color of the display can get out of step with the `minutesOffset` value. This version is in the example folder **Eg 06 No Globals Clock**

Host a website on GitHub

You now have something you might like to show off to the world. What better way to do this than putting it on a website for everyone to see? Then anyone who wants a time travelling clock can just go to your site and start it running. One way to do this is to create a website repository on GitHub. To do this you must have a GitHub account. You can only host one website on your account, but the site can contain multiple pages with links between them. We are going to start with a really simple web site which just contains the time travel clock we have just made. If you want to experiment with styles you can change the color, size and font of the text in the clock. In the next chapter we'll discover how to add images to pages too.

Make Something Happen

Host a web page on GitHub



Above you can see the endpoint of this exercise. This shows the clock program we have just created running in a website hosted by GitHub. Covering the steps to get this would take more pages than would fit in this chapter. You need to create an empty repository, use Visual Studio Code to clone the repository onto your PC, add the clock files to the repository, check in the changes and then synchronize the changes from the PC up to the repository. Then you just need to configure GitHub to tell it which part of the repository to share on the web and you have your new web page. The good news is that you won't be doing this very often. The better news is that I've made a step-by-step video that you can watch that takes you through the process.

Authors note: I tried to make a guided section for this, but it got much too large and convoluted. My thinking is that if they want to do this they'd be happy to watch a video.

What you have learned

This has been a very busy chapter. We've covered a lot of ground. Here is a recap plus some points to ponder.

- Git is a tool that was created to make it easier to work on large projects. It organizes units of work into repositories. A repository is a folder full of files plus a special folder managed by the git tool to track changes by making copies of changed files. You "commit" changes to the repository at which a snapshot is taken of their contents. You can return to the snapshot at any time. You can also compare the snapshot with the current

files.

- Git can be used on one machine by one person, or a git server can be set up for network access to repositories by several people. Git also provides a means of resolving changes to the same file by multiple people.
- GitHub is a cloud-based service that hosts git repositories. Users can take copies of (clone) repositories, work on them and then check them back over the network. A repository can be private to a particular user or public. Public repositories are the basis of open-source projects which have managers to accept contributions and then commit them after testing. A GitHub repository can be exposed as a web page, making GitHub a good way to host a simple web site.
- Git (and by extension GitHub) support can be added to software tools which can then make use of repository storage and management. The Visual Studio Code integrated development environment (IDE) works in this way. We can check repositories in and out as we work on them.
- Visual Studio Code also provides an extension mechanism which can be used to add extra features. The “Live Server” extension allows you to deploy websites on your PC so that you can test them.
- A web page is expressed by a file of text containing Hypertext Markup Language (HTML). The contains elements on the page that will be drawn by the browser when the page is displayed. The names of the elements are distinguished from text to be displayed by the use of < and > characters to delimit the element names, for example <head> is how the start of the header element would be expressed. The end of the header is expressed by an element containing the name preceded by a forward slash: </head>. The
 element (line break) does not need a corresponding </br> element.
- HTML elements can be nested. The <body> element contains all the elements which are to be displayed in the body of a web page.
- Html elements can have attributes which give information about the element. Attributes are added as named values in the definition of the element. The HTML <p id="timePar"> marks the start of a paragraph. This paragraph has an id attribute which is set to the value timePar.
- Elements in an html document can be given a class attribute which maps back to a style definition in a stylesheet file which is loaded by the browser. The source HTML file specifies the stylesheet file location using a link element to the head of the document.

- The browser uses the HTML file to create a Document Object Model (DOM). The DOM is a software object that describes the web page structure. The DOM contains references to objects that represent the elements described in the HTML Page.
- Element objects in the DOM contain property values which are mapped onto the attributes assigned in the HTML. For example, the `class` attribute on an element in an HTML document is mapped onto the `className` property in the element object in the DOM. This makes it possible for JavaScript programs to change the values of properties and their appearance on the page. We used this ability to change the color of the text in the time travel clock.
- You can use the Elements tab in the Developer Tools for the browser to look at the elements in the DOM.
- An element in an HTML document can be given an id attribute that allows a JavaScript program to find it in the DOM. The method provided by a document that will get an element by its id is called, not surprisingly, `getElementById`. If the method can't find an element with the requested id it will return a null reference.
- You can change the `textContent` property of a paragraph element by assigning a string to it. The element will then display this new text on the web page.
- Some HTML elements can generate events. An event is identified by a name (usually starting with the word on) and contains a string of JavaScript code to be performed when the event occurs. The `body` element can have an `onload` attribute that specifies JavaScript to run when a web page is loaded by the browser.
- A web page can contain button elements that generate events when the user clicks on them. When the user of the web page clicks the button a JavaScript function can be called to respond to this event.
- GitHub can be made to host web pages as well as store repositories.

To reinforce your understanding of this chapter, you might want to consider the following "profound questions".

What is the difference between Git and GitHub?

Git is the program that you run to manage your repositories. GitHub is a cloud based service that hosts repositories and is accessed using the git program.

Can I use Git on my own machine?

Yes you can. You can use it to manage a repository stored on your machine. You can also set up your own Git server and use it to store private projects on your home network.

Will I ever use the git program directly?

I try to avoid doing this too much. Most of the time Visual Studio Code will hide a lot of the complexities of git, but every now and then, particularly if you work in group projects, you might find that you have to type in a git command to fix something.

Are there different kinds of Open-Source projects?

Yes there are. This is controlled by the terms of the license agreement assigned to a particular project. It is worth reading through these to find out what permissions you can give people on the projects that you make open source. You should also check when you use an open-source project that you are not breaking any of their conditions.

Why is HTML called a markup language?

HTML is used to express the design of a view of a web page. In the days before computers a printer would write instructions on the original copy of a document they were printing. The instructions would specify the fonts and type sizes for the text to be printed. This was called “marking up”. HTML can be used to express how text is to be formatted and so it was given the name markup.

What would happen if I just gave a browser a file of text?

The browser tries very hard to display something, even if it doesn't look like a proper HTML document. It would display a file of text as a single line however, as it ignores line feeds and reduces multiple spaces to 1. If you want to display separate lines and paragraphs you have to add the appropriate formatting elements.

Where does the browser store the Document Object Model?

The Document Object Model is stored in the memory of the computer by the browser when a web page is loaded. A software object provides data storage and methods that are used to interact with the data stored in the object. The HTML file sets the initial elements in the object and their initial values.

What is the difference between HTML and HTTP?

HTML is a way of expressing the content of a web page. HTTP (Hypertext Transport Protocol) is all about getting that page from the server to the browser. The browser uses the HTTP protocol to get resources and the server delivers them. The browser sends an HTTP get request (which actually includes the word GET). The server finds the resource and sends it

back. A get request always returns a status value. A status of 200 means “all is well, here is the data” whereas 404 is the infamous “file not found” error that has become so notorious that it now appears on T-shirts.

Is HTML a programming language?

No. A programming language expresses a solution to a problem. Solving a problem may involve making decisions and repeating behaviors. HTML does not have constructions for doing either of these things. HTML is used to express the contents of a logical document, not to solve problems.

What happens if a JavaScript program changes a visible property on an element very quickly?

We have seen that a JavaScript program can display messages in the browser by updating a property on an element in the document object. We set the `textContent` property on a paragraph to update the clock. If we do this very quickly the browser will not be able to keep up. The browser updates at a particular rate, usually 60 times a second. There is a way that code in a web page can get an event each time that the browser wants to update the page. We will be using this in the last chapter of this text when we look at games.

Does every element on a web page need to have an id attribute?

We added an `id` attribute to the time paragraph on the web page so that the clock program could find that paragraph and change the text on it when the clock ticks. We only need to add an `id` to the elements that the JavaScript code needs to find.

What happens if an event function gets stuck?

Events can be assigned to JavaScript functions by attributes in web pages. We have seen how the `onload` attribute of the `body` element lets us call a JavaScript function when a page loads. But what happens if the function never returns? We’ve all had that experience where you visit a web site and your browser stops. This can be caused by a stuck JavaScript function which is triggered by an event in the web page. If you write a function that never returns (perhaps because it gets stuck in a loop) and then call this from an `onload` attribute you will find that the web page will never complete loading. If a function called from `setInterval` never returns the web page will not stop immediately, but it will progressively slow down as the computer memory fills up with more and more untermiated processes. If a JavaScript function assigned to a button gets stuck the user will not be able to press any other buttons until that function returns.

What is the web address of a page hosted by GitHub web server?

It is an address made from the GitHub username and the name of the repository that holds the site files. However, if you register your own domain name you can configure GitHub to use it. In other words, you can set the address of the page to any domain that you can get hold of.