

Copyright © 2018 Kenneth Cosh.

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review.

ISBN: 978-616-398-352-7

Printed by Kenneth Cosh in Thailand. Published by Chiang Mai University.

First printing, 2018.

Chiang Mai University 239 Huay Kaew Road. Chiang Mai, 50200

www.kencosh.com

Preface

This book provides an introduction to Web Programming languages, beginning with the basics of Hypertext Markup Language (HTML). No previous experience is assumed, although some basic programming concepts would be an advantage. The target audience is for second year undergraduate students within a computing related discipline, although the book should be accessible for any readers.

The book is broken into four parts, beginning with the basics of static webpages. As well as HTML the first part covers Cascading Style Sheets (CSS), which are used for styling a webpage and positioning elements on it. The second part focuses on Server-side scripting, where the WAMP stack is used (Windows, Apache, MySQL and PHP). This part uses PHP to power the server and connect with a MySQL database. Part three deals with Client-side scripting, covering JavaScript. As well as introducing pure JavaScript, the JQuery library is discussed with its tools for handling browser events and manipulating the DOM (Document Object Model). Asynchronous JavaScript is also covered, allowing different elements in the user interface to be loaded separately. The final part covers modern web development technologies, where first the features of HTML5 are introduced. Sometimes today JavaScript is being used everywhere, on the server as well as on the client. Here Node.js is introduced to power a server, as part of the MEAN development stack (MongoDB, Express.js, Angular and Node.js). Express.js is a lightweight framework for Node.js on the server. With modern data demands, MongoDB is introduced as an alternative database. Finally Angular is introduced, an increasingly popular front-end framework.

Throughout the book there are code examples to demonstrate the concepts discussed, so readers are encouraged to practice their skills as they take their first steps towards becoming a Web Developer.

Table of Contents

Part 1 :- Foundations of Static Webpages (HTML & CSS)	1
Chapter 1 - Introducing HTML	2
1.1 Basic HTML Tags	5
1.2 HTML Tables	7
1.3 Images and HTML Attributes	8
1.4 Hypertext & Links	9
1.5 HTML Comments	10
Chapter 2 - CSS for Style	12
2.1 Why Use CSS?	13
2.2 Introducing CSS	14
2.3 Selectors	17
2.4 & <Div>	19
2.5 Where to CSS?	19
2.6 More on Selectors	20
2.7 CSS Properties	21
Chapter 3 - CSS for Layout	27
3.1 The Position Property	29
3.2 Float & Clear	32
3.3 Borders, Margins & Padding	35
3.4 A Simple CSS Image Gallery	38
3.5 A Simple CSS Hover Box	39
Part 2 :- Server Side Scripting with PHP	43
Chapter 4 - Introducing PHP	44
4.1 Setting Up A Webserver	45
4.2 PHP Basic Syntax	46
4.3 PHP Variables	47
4.4 PHP Operators	49
4.5 PHP Flow Control	50
4.6 PHP Form Validation Example	52
Chapter 5 - PHP Functions and Objects	58
5.1 Array Functions	59
5.2 Mathematical Functions	64
5.3 Date and Time Functions	67
5.4 Defining Your Own Functions	68
5.5 Include and Require	69
5.6 Objects in PHP	70
5.7 Creating a Calendar	71

Chapter 6 - PHP Files & MySQL Databases	77
6.1 PHP and Files	78
6.2 Introducing MySQL and phpmyadmin	81
6.3 Structured Query Language	84
6.4 MySQLi	87
Chapter 7 - Cookies, Sessions & Security	92
7.1 Cookies	93
7.2 Sessions	95
7.3 PHP Login Script	95
7.4 Online Security Threats	103
Part 3 :- Client Side Scripting with JavaScript and JQuery	105
Chapter 8 - Introducing JavaScript	106
8.1 Variables	108
8.2 Operators	110
8.3 Control Statements	111
8.4 Arrays	113
8.5 Functions	115
8.6 getElementById()	116
8.7 Form Validation Example	118
Chapter 9 - Introducing JQuery	122
9.1 Handling Events with JQuery	125
9.2 JQuery Effects	129
Chapter 10 - JQuery and the DOM	137
10.1 The Document Object Model	138
10.2 Manipulating the Document Object Model	140
10.3 Navigating the Document Object Model	145
Chapter 11 - Asynchronous JavaScript and JQuery UI	155
11.1 Introducing AJAX	156
11.2 JQuery and AJAX	158
11.3 JQuery UI	160
Part 4 :- Modern Web Development Technologies	167
Chapter 12 – HTML 5	168
12.1 New Tags (and Deprecated Tags)	169
12.2 Audio & Video	172
12.3 The Canvas	173
12.4 Geolocation	177
12.5 Local Storage	178
12.6 Web Workers	180

Chapter 13 – Introducing Node.js	183
13.1 PHP vs Node.js	184
13.2 Setting up Node.js	185
13.3 Events in Node.js	189
13.4 Node Package Manager (NPM)	190
13.5 Sending Emails with Node.js	192
Chapter 14 – Node.js, MySQL & MongoDB	194
14.1 Node.js & MySQL	195
14.2 Node.js & MongoDB	199
14.3 MySQL vs MongoDB	205
Chapter 15 – Express.js	207
15.1 Getting Started with Express.js	208
15.2 Routing with Express.js	209
15.3 Creating an API with Express.js	211
15.4 Uploading Files with Express.js	215
Chapter 16 – Angular	218
16.1 Getting Started with Angular	220
16.2 Components in Angular	222
16.3 Adding a TypeScript Class	224
16.4 Pipes and Two-Way Data Binding in Angular	227
16.5 Event Binding in Angular	228
16.6 Services in Angular	229
16.7 Routing in Angular	233

List of Tables

1.1	Key HTML Tags	4
1.2	Style Related HTML Tags	5
2.1	Text Styling Properties	22
3.1	The Four Positioning Properties And Their Effects	32
4.1	String Related Functions in PHP	49
4.2	Arithmetic Operators in PHP	49
4.3	Assignment Operators in PHP	49
4.4	Logical Operators in PHP	50
5.1	Sorting Arrays in PHP	64
5.2	Trigonometry Functions in PHP	66
5.3	Parameters for the PHP date() Function	68
6.1	PHP File Open Modes	78
8.1	Arithmetic Operators in JavaScript	110
8.2	Assignment Operators in JavaScript	110
8.3	Logical Operators in JavaScript	110
8.4	Array Functions in JavaScript	116
9.1	Events that can be Handled by JQuery	125
9.2	Fade Effects in JQuery	131
9.3	Slide Effects in JQuery	132
10.1	Adding Elements to a Page with JQuery	142
10.2	Traversing Across the DOM	150
11.1	The readyState Property of XMLHttpRequest	158
12.1	HTML5 Semantic Elements	171
14.1	MongoDB Query Comparison Operators	204

List of Figures

1.1	My First Webpage	4
1.2	My Second Webpage	5
2.1	A CSS styled table	14
2.2	A Simple CSS Style Example	17
2.3	An Example Of A Class Selector	18
2.4	An Example Of ID Selectors and Layers	19
3.1	Basic CSS Positioning	29
3.2	Static And Relative Positioning I	30
3.3	Static And Relative Positioning II	30
3.4	Relative And Absolute Positioning	31
3.5	Problem With Relative Positioning	33
3.6	An Example Of Float	34
3.7	The Clear Property	35
3.8	Different Border Styles	35
3.9	The Box Model	36
3.10	A Simple CSS Image Gallery	39
3.11	A Simple CSS Hover Box	40
4.1	A PHP Request	45
4.2	Hello World in PHP	46
4.3	A Simple Form	53
4.4	A Welcome Page using \$_POST	53
4.5	A Welcome Page using \$_GET	54
4.6	The POST Array	55
5.1	print_r() in PHP	60
5.2	array_count_values() in PHP	60
5.3	array_diff() in PHP	61
5.4	array_intersect() in PHP	61
5.5	array_flip() in PHP	62
5.6	array_reverse() in PHP	62
5.7	shuffle() in PHP	63
5.8	Sorted Array in PHP	64
5.9	All 6 Different Types of Sort in PHP	64
6.1	PHPMyAdmin Menu	82
6.2	Creating a Database in PHPMyAdmin	82
6.3	Creating a Table in PHPMyAdmin I	83
6.4	Creating a Table in PHPMyAdmin II	83
6.5	Table Structure in PHPMyAdmin	84
6.6	Browsing a Table in PHPMyAdmin	85
6.7	An Updated Table in PHPMyAdmin	86
6.8	The Results of fetch_array()	89
6.9	The Results of fetch_assoc()	90
6.10	The Results of fetch_object()	90
8.1	Simple Form for JavaScript Validation	118
9.1	Opacity of Page Elements	131
10.1	Sample Page to Illustrate the DOM	139
10.2	Example of the DOM	139
10.3	Textual Version of the DOM	140

10.4	Demonstrating before(), prepend(), append() and after() in JQuery	143
10.5	Editing the Box Model with JQuery	144
10.6	Sample age for Demonstrating DOM Navigation	146
10.7	The parent() Function in JQuery	146
10.8	The parents() Function in JQuery	147
10.9	The parentsUntil() Function in JQuery	147
10.10	The children() Function in JQuery I	148
10.11	The children() Function in JQuery II	149
10.12	The find() Function in JQuery I	149
10.13	The find() Function in JQuery II	150
10.14	Traversing Using Filters	151
10.15	The first() Function in JQuery	151
10.16	The last() Function in JQuery	152
10.17	The eq() Function in JQuery	152
10.18	The filter() Function in JQuery	153
10.19	The not() Function in JQuery	153
11.1	How AJAX Works	157
11.2	The Standard HTML Date Picker, as seen in Chrome	161
11.3	The Standard HTML Date Picker, as seen in Edge	162
11.4	JQuery UI Date Picker	163
11.5	JQuery UI Simple Dialog Box	164
11.6	JQuery UI Draggable and Droppable	164
12.1	Sample Layout of Semantic Elements	170
12.2	The Color Input Type	171
12.3	The Range Input Type	172
12.4	Audio Playback Controls	173
12.5	Video Playback Controls	173
12.6	Simple Line Drawn onto Canvas	174
12.7	The Japanese Flag drawn onto Canvas	175
12.8	The Thai Flag drawn onto Canvas	176
12.9	A Gradient Using the Canvas	176
12.10	Google Map Powered by Geolocation	178
12.11	Local Storage	179
12.12	A Simple Web Worker	181
13.1	Initiating Node.js in the Command Prompt	186
13.2	Hello World by Node.js	186
13.3	Extracting the URL in Node.js	187
13.4	Parsing the URL in Node.js	188
13.5	A Console Log Event in Node.js	189
13.6	Sample Module from NPM	191
13.7	The num2fraction Module	191
13.8	Demonstrating the num2fraction Module	192
14.1	The empty Calendar Database Schema shown in MySQL Workbench	196
14.2	The appointments Table Structure as seen in MySQL Workbench	197
14.3	Querying the appointments Table in MySQL Workbench	198
14.4	Outputting the Results of a Query	198
14.5	Inspecting the MongoDB Database using Compass I	201
14.6	Inspecting the MongoDB Database using Compass II	202
15.1	Hello World using Express.js	209
15.2	Returning Parameters as JSON	210

15.3	Returning Request Parameters	212
15.4	Returning All Appointments	213
15.5	API Returning Appointments for Specified Month	213
15.6	Result of API to Insert Appointment	215
15.7	Uploading Files with multer	216
16.1	Basic View of AngularJS MVC Architecture	219
16.2	The default Angular App	221
16.3	Hello World in Angular	222
16.4	Simple Child Component in Angular	224
16.5	Simple Child Component displaying a TypeScript Object	225
16.6	Displaying a List Using *ngFor	226
16.7	Two Way Binding in Angular	227
16.8	Event Binding in Angular	228
16.9	England Players in MongoDB	229
16.10	England Team List View	235
16.11	England Player View	236

Part 1

Foundations of Static Web Pages (HTML & CSS)

Chapter 1

Introducing HTML

Chapter 3

CSS For Layout

Chapter 2

CSS For Style

Part one deals with the fundamentals of Web development, namely HTML and CSS. HTML (or Hypertext Markup Language) is one of the key cornerstone languages of the web used to markup hypertext, instructing the browser how to display the various elements of a webpage. CSS (or Cascading Style Sheets) is another of the key technology used to style the HTML elements on each page. CSS allows the separation of content and presentation with CSS providing the instructions for the style of each element and also the layout of each page.

Chapter 1

Introducing HTML

Objectives

This chapter introduces Hyper Text Markup Language or HTML, the core language used to make basic web pages. After reading the chapter you should;

- Understand how tags and text are used to create simple webpages
- Know the key tags defined within the HTML language
- Be able to create simple, static webpages that include
 - Text
 - Images
 - Tables
 - Links
 - Basic styling

Contents

- 1.1 Basic HTML Tags
- 1.2 HTML Tables
- 1.3 Images and HTML Attributes
- 1.4 Hypertext & Links
- 1.5 HTML Comments

HTML, or Hypertext Markup Language is the standard markup language used to create webpages. Hypertext is simply text that is not linear – while traditional text, such as in this book, is constrained in a sequence, hypertext instead can contain links allowing readers to travel through it in a non-linear sequence. Unlike some of the other languages we will explore in this book, HTML isn't a programming language, it is simply a markup language, used to "markup" hypertext. Hypertext can be marked up in many ways, essentially instructing a web browser how to display the page.

HTML isn't a programming language, it is simply a markup language, used to "markup" hypertext

HTML consists of text and tags, with the tags instructing the browser what to do with the text – while the text will appear on the webpage, the tags won't. The tags are enclosed between angle (or pointy) brackets, such as <html>. Often tags come in pairs with an opening and closing tag – the closing tag contains a slash, such as </html>. Everything in between the opening and closing tag is treated according to which tag has been used – in this case everything between the opening tag <html> and the closing tag </html> is to be treated as html – the HTML tag is used to surround the whole webpage. Note that tags are not case sensitive, you could use <HTML> or <html>, but it is recommended to use lower case.

Let's dive straight in and create our first webpage! As html is simply text, we can use any text editor to create a webpage. You can use "Notepad" just make sure you save the file with the extension ".htm" or ".html". Alternatively you could use a text editor like Dreamweaver or Sublime Text, these have an advantage that they use colours to differentiate between different tags and text, which makes it easier for debugging, and they can also highlight errors. Go ahead, open your text editor and type in the following and save the file as index.htm.

```
<html>                                                                    index.htm
<head>
  <title>My First Webpage</title>
</head>
<body>
  Hello World!
</body>
</html>
```

This is the classic first webpage. If you open this file using your browser you should see something similar to this;

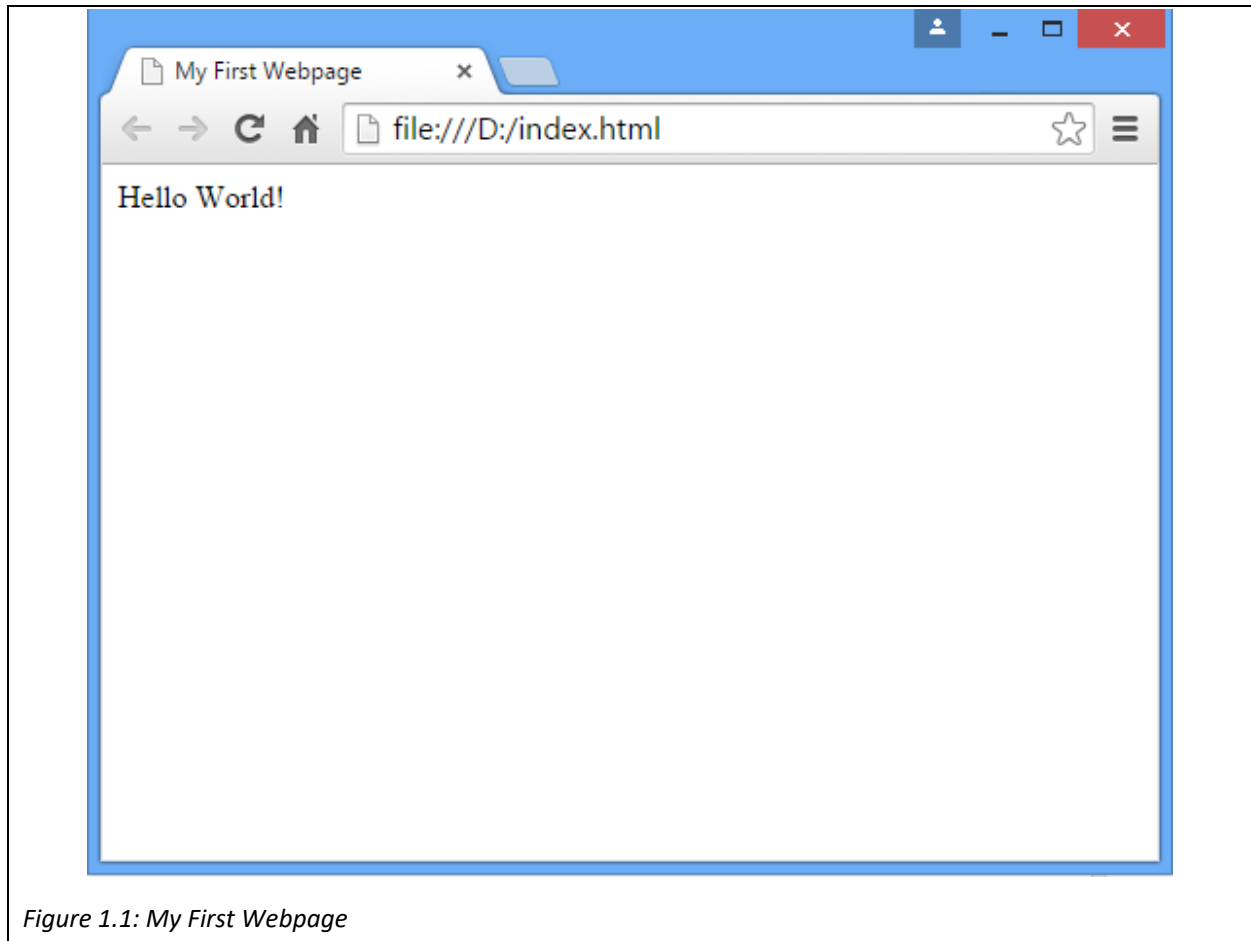


Figure 1.1: My First Webpage

The only text that appears on the webpage is “Hello World!”. The rest of the html file is defining how the browser should present it. It consists of 4 important tags;

<html>...</html>	The HTML tags surround the rest of the page, indicating that everything insider is html.
<head>...</head>	A webpage is divided into 2 sections, a head and a body. The HEAD tags can contain a variety of header data – things which generally don’t appear on the website. We will later see how javascript or cascading style sheets can be included in the header.
<body>...</body>	The BODY tags contain the part of the webpage that you can see in the browser window, in this case the text “Hello World!”.
<title>...</title>	The title is part of the header and determines what appears in the browser toolbar. The title is also used when a page is bookmarked and when a page is displayed in a search engines results.

Table 1.1: Key HTML Tags

The <html>, <head> and <body> tags are important and are the general structure of webpages. You can see them if you view the source html of any page you visit.

Quick Tip

You can view the source of any webpage in your browser. In Chrome you can access the source using “Ctrl-U”, or by right clicking somewhere on the page and choosing “View page source”. If you look at your first webpage you will see the same as you typed in your text editor. You can see the source of any other page, which is great for learning how other web developers have done things. This may be complicated for now, but it is a useful tool and you should see the <html>, <head> and <body> tags on any other page.

1.1 Basic HTML Tags

HTML was originally specified by Tim Berner’s Lee in 1990 while he was working at CERN with the intention of allowing colleagues to share documents. The earliest version of HTML consisted of just 18 tags, but as the internet has evolved, the language has been developed through various versions until the latest version; HTML5. Some of the original tags have remained the same, while others are no longer supported. Some HTML tags can be used to define the style in which things appear on a page, and these elements have in many cases been replaced by cascading style sheets as we will see in chapters two and three. We can use some basic HTML tags to make our webpage look a little more interesting.

<p>...</p>	P is for paragraphs, and is used to define a text into paragraphs. By default the browser will add some space (or a margin) before and after each <P> tag.
<h1>...</h1> <h2>...</h2> <h3>...</h3> <h4>...</h4> <h5>...</h5> <h6>...</h6>	A variety of headings can be easily added to a page using the tags from <H1> to <H6>. These tags highlight text into different sizes depending on which level heading is used.
... <i>...</i>	 can be used to stress emphasis, or to highlight a piece of text, by default making it appear like italics.
... ...	Strong tags can be used to highlight important text, or make it appear bold.
 	 is used to create an unordered list of something, and the list will consist of list items

Table 1.2: Style Related HTML Tags

Most of these tags are used to define the style of how text appears on a webpage, and today style is controlled by cascading style sheets as we shall see in chapters two and three. Nonetheless these tags can be used to make a webpage appear more interesting. Note that for most browsers the effect of and <i> is

the same, and similarly the effect of `` and `` is equivalent. There is however a subtle difference; `` or 'bold' defines how the text should appear, whereas `` simply indicates that the selected text is more significant than the surrounding text. Similarly while `<i>` defines that the text should be italic, `` indicates that it should be emphasized. For most browsers this difference isn't noticeable, but if we consider a browser for a blind person, the visual appearance of the text is not significant, similarly some browsers on mobile devices may display the text differently. We can use these tags to create a slightly more interesting webpage.

```
<html>
<head>
  <title>My Second Webpage</title>
</head>
<body>
  <h1>Heading 1</h1>
  <h2>Heading 2</h2>
  <h3>Heading 3</h3>
  <h4>Heading 4</h4>
  <h5>Heading 5</h5>
  <h6>Heading 6</h6>
  <p>This is an <em>interesting</em> paragraph of text with an <strong>important</strong> word.</p>
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherry</li>
  </ul>
</body>
</html>
```

If you look at this page in a browser you should see different sized headings, and a bulleted list of fruits. Notice how some html tags are nested inside other tags; in other words the `` tag is nested inside the `<p>` tag, which is in turn nested inside the `<body>` tag, similarly the `` tags are nested inside the `` tag. Always be careful to close your tags correctly – if an element is nested inside another element, make sure to close the nested element before closing the parent element. Not all elements have a closing tag, for example the `
` tag is used to put in a line break and it needs no closing tag, although it can be closed by using `
`.

*Not all elements have closing tags, for example the `
` tag is used to put in a line break and it needs no closing tag, although it can be closed by using `
`.*

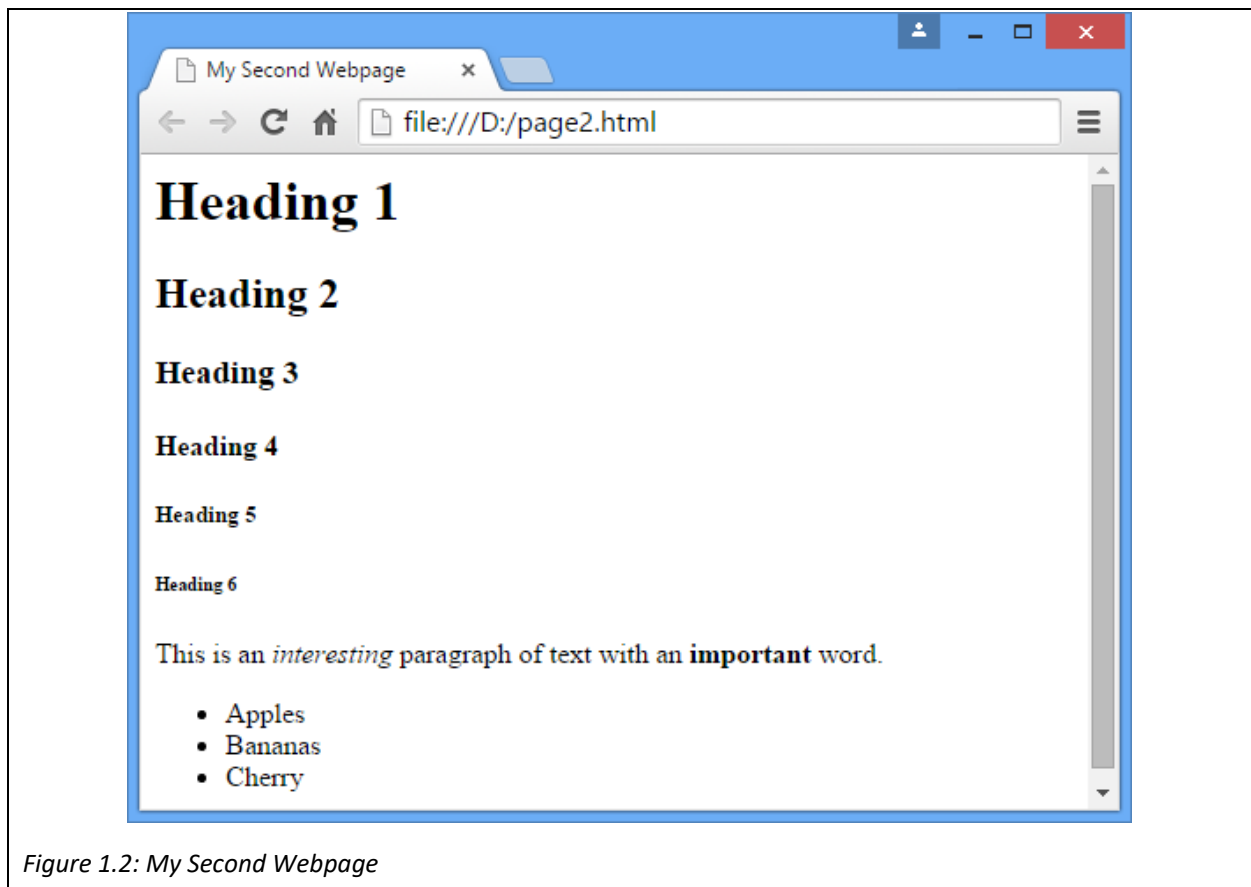


Figure 1.2: My Second Webpage

1.2 HTML Tables

HTML also includes several tags for creating tables. Tables are similar to spreadsheets and are useful for displaying data in two dimensions.

```
<table> table.htm
<tr>
  <td>Country</td>
  <td>Capital</td>
  <td>Population</td>
</tr>
<tr>
  <td>Thailand</td>
  <td>Bangkok</td>
  <td>67 Million</td>
</tr>
</table>
```

A table consists of the <table> tag. Nested inside the <table> tag are each table row is defined using the <tr> tag. Nested inside each row the columns are defined using table data, the <td> tag. The preceding example would show a simple table with 2 rows, one with the column headings, and one with some data about Thailand. As yet the table wouldn't look very interesting, as we will need to investigate cascading style sheets to make the table look better. Further table related tags include <thead>, <tbody> and <tfoot>, which can be used to indicate where a table has a heading row, a body and a footer. For many years it was common practice to use tables to lay out a webpage, but a modern approach uses CSS instead. While tables can still be used for laying out the different parts of a webpage, many of the attributes of tables are now deprecated.

Quick Tip

Don't use tables for layout!

- Tables often use more bytes of markup, making pages take longer to download.
- The browser needs to download most of a table before it can render any of the page.
- Tables can make website maintenance and particularly redesign much more complicated.
- Tables can make content inaccessible to screen readers.
- Tables can break text copying.

In short, tables weren't meant for page layouts, but they ARE meant for displaying tabular data!

1.3 Images and HTML Attributes

Webpages look much better with images and adding images to a page is easy, using the tag, short for 'image'. The tag is a little different from the tags we have encountered so far. Firstly an tag is another tag where there is no closing tag, as we want to place an image on a page, but we don't need to indicate when we have finished placing the image on the page. Secondly, an tag needs some attributes, or parameters, most importantly the location of the image you want to place on the page. The first key attribute is the "src" attribute, short for source which indicates the source of the image file. The following would display an image called 'picture.jpg' in the same directory as the html file.

```

```

It is a good idea to organize the various parts of a webpage within your working directory, and that includes images. Standard, good practice will store images in a separate folder, perhaps named '/img' or '/images'. The src attribute can then be a *relative link* to the file, in this case;

```

```

A *relative link* refers to a URL (Uniform Resource Locator) in relation to the page that is currently being looked at. In this case the resource being located is the image, which is stored in the img folder. An alternative to relative links is to use an *absolute link*. Absolute links provide the full location of the resource, including the protocol used to get the resource and the server on which it is stored, for example;

```

```

There are different arguments for using either relative or absolute links, relative links are obviously shorter, but Search Engine Optimizers (SEOs) may argue to use absolute links rather than relative links to avoid issues like duplicate content, but while in the future this will be important when working on a live webserver, for now it is ok to use relative links if you are working on your local machine.

A Relative Link refers to a URL in relation to the page that is currently being looked at, while an absolute link provides the full location of the resource including the protocol used to get the resource and server on which it is stored.

The tag can also take other attributes, with 2 particularly useful attributes, 'alt' and 'style'. The attributes are set inside the image tag in much the same way as the 'src' attribute. The 'alt' attribute is used to supply alternative text for an image – consider if a blind visitor views your page, or perhaps someone's connection is particularly slow. The contents of the alt attribute are also displayed when the page visitor hovers their mouse above the image. In this case the text in the alt attribute can be presented instead, it may also be useful for SEO. The size of the image can also be specified using the style attribute. We will investigate the style attribute further in coming chapters when looking at cascading style sheets, but for now;

```

```

The 'alt' attribute is used to supply alternative text for an image

1.4 Hypertext & Links

As mentioned at the beginning of this chapter the key to the web is that it is written using hypertext, and hypertext allows visitors to jump from place to place across the internet. While traditionally text was organized in a linear fashion, which lead to books being organized and arranged on shelves, such as in a library. Text here is constrained to only allow one word to follow another, or one sentence to follow another sentence. Hypertext is different, hypertext allow the reader to take different routes through the information, by following hyperlinks.

Adding hyperlinks to a webpage is easy, it just involves using another tag, this time the <a> or anchor tag. The anchor tag has an opening and closing tag, which are around the text, (or image), that you want to form the link. The anchor tag needs attributes, and for a link the most important parameter is the url that the link links to. This is stored in the 'href' attribute.

```
<a href="http://www.example.com">link here</a>
```

The href attribute can take an absolute link, or a relative link, just the same as for the source of an image. With default styling, the words “link here” will appear underlined in blue, and when clicked the browser will load the page stored at <http://www.example.com>. Adding links to your page does of course mean that your visitor may leave your site, so another useful attribute is the target attribute. The target attribute specifies where the url should open; to get the link to open in a new window specify “_blank” as the target attribute.

```
<a href="http://www.example.com" target="_blank">link here</a>
```

Links needed link to the top of a webpage, often it is useful to link to a particular part of a page, or sometimes link to a different part of the same page. For this we first need to create an identifier for where the link should direct to. This is done using the “id” attribute which can be added to most of the html tags we have discussed so far. The “id” attribute is particularly important for many reasons that we will see later in the book, but for now the we could create a heading which has an “id” attribute.

```
<h1 id="html-linking">Links in HTML</h1>
```

The “id” attribute isn’t displayed in the browser window, but it does uniquely identify this particular “h1” tag. Ids should only appear once on a page, so that they are uniquely identifiable. A link can then be created to this particular part of the page using a “#”, as follows;

```
<a href="#html-linking">Go to the HTML Linking Section</a>
```

1.5 HTML Comments

When writing code in any programming language it is often useful to add comments. Comments can be used to send messages to other developers who might look at your code, or messages to yourself in case you forget why you have written it in a certain way. Comments are also very useful for debugging as you can remove parts of your code while testing it. Comments in HTML are contained within the comments tag.

```
<!--This is a comment -->  
<!--  Don't show this yet! -->
```

Key Points

- HTML (Hyper Text Markup Language) is used to create simple webpages.
- An HTML page consists of text and tags, with tags instructing a browser what to do with the text.
- Opening tags, such as `<html>`, are matched with closing tags, such as `</html>`.
- Some HTML tags are used to define the structure of the document, such as `<head>` and `<body>`.
- Some HTML tags instruct the browser how text should be displayed, such as `` or `<i>`.
- The table tags can be used to display tabular data on a page.
- Pictures can be included on a webpage using the `` tag, which has no closing tag.
- Some tags, such as the `` tag need attributes, or parameters, to give the browser more information such as how big the image should be, or where the file is located.
- The anchor tag, `<a>`, is used for creating links.

Further Resources

- 1) The w3schools website has a lot of excellent resources for web developers. Their default HTML page can be found here:-

<https://www.w3schools.com/html/default.asp>

and their HTML reference guide can be found here:-

<https://www.w3schools.com/tags/default.asp>

- 2) Codecademy offers a course demonstrating fundamentals of HTML combined with CSS (as we will explore in chapters 2 and 3). Their course can be found here:-

<https://www.codecademy.com/learn/learn-html-css>

- 3) html.com has a reference guide about each of the key tags, and tutorials on several of the topics discussed in this chapter. Their page can be found here:-

<http://html.com/>

Assignment

You now have all the skills you need to dive in and create your first webpages! Create a simple website to introduce yourself to the online world. Your site should have at least 3 pages, linked together, contain pictures and use a table to show your weekly timetable.

Chapter 2

CSS for Style

Objectives

This chapter introduces Cascading Style Sheets (CSS), a language used to define the presentation of an HTML document. This chapter discusses how CSS is used to control the style of a page or website, while the following chapter discusses using CSS to control the layout of a page. After reading this chapter, you should;

- Understand the importance of separating the style of a page from its content and how CSS enables developers to do that.
- Be able to define selectors (HTML, class and id selectors) assigning different properties and values.
- Know how to use span and div tags to implement styles.
- Understand the different places CSS can be added to a webpage.
- Be familiar with some of the key properties that can be controlled by CSS.

Contents

- 2.1 Why Use CSS?
- 2.2 Introducing CSS
- 2.3 Selectors
- 2.4 & <Div>
- 2.5 Where to CSS?
- 2.6 More on Selectors
- 2.7 CSS Properties

CSS, or Cascading Style Sheets, is a language used to define the presentation of a document written in mark up language, such as HTML. The main purpose of CSS is to separate the content of a document from aspects of its presentation, such as layout, colors and fonts. Separating the style from the content had numerous advantages, and is a common design philosophy. One clear advantage to using CSS is reduced complexity, when all the presentation instructions are stored in a different file. CSS can reduce repetition, and potential mistakes, as different documents use the same styles, keeping consistency across an entire website, and making changes to the style of a website much easier – consider if your site has 100 pages, and you needed to make a simple design change, without CSS you may need to edit all 100 pages, but with CSS the whole site can be changed by editing a single line. For these reasons, CSS has become a cornerstone language, core to web development. In this chapter we investigate using CSS to control the style of a webpage, and in the following chapter we look at using CSS to control the layout of a webpage.

The main purpose of CSS is to separate the content of a document from aspects of its presentation, such as layout, colors and fonts

A style is a definition of the form something takes – for text this could include the color, the font used, the size and other stylistic aspects such as italics, bold or underline. For other elements such as an image the style may include the width, height and position. Styles may also include presentational aspects such as a border or the surrounding padding. Each style is a combination of these aspects, and each style is given a unique name – known as a selector. The selectors and their corresponding styles are then defined in one place – normally in the cascading style sheet. HTML tags then just refer to a particular selector when they need to activate a particular style.

2.1 Why Use CSS?

There are several good reasons for using CSS, beyond conforming to the design philosophy of separating form and content. Using CSS allows you to define all the styles in one place – rather than needing to repeat the styling information many times across many pages. To have a consistent design across a multiple page website, using only HTML you may need to continually specify the same styling information – also consider if you need to change that style in the future, with CSS you only need to make one change in one place to change the look of all the pages. CSS also means you are no longer limited to the standard HTML tags, such as <P> and <H1>, with CSS you can create your own selectors precisely defining the style with the same accuracy as in a word processor. With CSS you can completely redefine the effect of the standard HTML tags, and also redefine features such as links – they no longer need to be underlined and blue, and you can define different effects if the user hovers their mouse over a link. As we will see in chapter 3, CSS also allows you to place elements, such as images, onto the page with pixel precision, you can even define multiple layers to allow

elements to appear on top of other elements, which can be useful for drop down menus. Using CSS can also speed up page load times, improving the experience for your site visitors.

Advantages of using CSS
* Separation of form & content
* Define the look of pages in one place
* Easily change the look of pages
* Define more font attributes than the standard tags
* Position elements with pixel precision
* Redefine entire HTML tags
* Define custom styles for links (not just a blue underline)
* Define layers, so elements can appear on top of other elements

2.2 Introducing CSS

Let's look at a simple example of how you might set up a style. Suppose you want to make a simple table to display 2-letter country codes for different countries, but were tasked with styling that table as shown below. Here the first column shows the 2-letter country code, and the second column shows the country. Notice the two columns have slightly different fill shading, the first column's text is bold, blue and slightly larger than the second column which is italic.

uk	<i>United Kingdom</i>
us	<i>United States</i>
th	<i>Thailand</i>

Figure 2.1: A CSS Styled Table

Using the HTML covered in chapter 1, this is a fairly straightforward task, as illustrated in the code on the following page. A table is created and for each table data tag, <td>, various parameters are set. For the text in each cell the font tag, , carries further parameters defining the style of the text. It is worth noting that the tag is deprecated in the latest version of HTML, HTML 5, as the standard and much better way of coding this example is using css, as will be explored shortly. It is also worth noting that the html font tag is limited, particularly when it comes to specifying the exact size of text.


```

<table>
  <tr>
    <td bgcolor="#eeeeee" align="left">
      <font face="arial" size="4" color="blue"><b>uk</b></font>
    </td>
    <td bgcolor="#dddddd" align="center">
      <font face="arial" size="2" color="black"><i>United Kingdom</i>
    </td>
  </tr>
  <tr>
    <td bgcolor="#eeeeee" align="left">
      <font face="arial" size="4" color="blue"><b>us</b></font>
    </td>
    <td bgcolor="#dddddd" align="center">
      <font face="arial" size="2" color="black"><i>United States</i>
    </td>
  </tr>
  <tr>
    <td bgcolor="#eeeeee" align="left">
      <font face="arial" size="4" color="blue"><b>th</b></font>
    </td>
    <td bgcolor="#dddddd" align="center">
      <font face="arial" size="2" color="black"><i>Thailand</i>
    </td>
  </tr>
</table>

```

Notice that the same styling properties are repeated three times through the page. This can be resolved by setting up the style and calling it when needed. Assuming the styles "col1" and "col2" have been defined, they could then be applied as below.

```

<table>
  <tr><td class="col1">uk</td>
    <td class="col2">United Kingdom</td></tr>
  <tr><td class="col1">us</td>
    <td class="col2">United States</td></tr>
  <tr><td class="col1">th</td>
    <td class="col2">Thailand</td></tr>
</table>

```

There are various places to define styles, as we will shortly see, but the following definitions could be placed in the head section of the webpage.

```

<style>
.col1 {
    background-color:#eeeeee;
    text-align:left;
    font-family:Arial, Helvetica, sans-serif;
    font-size:18px;
    color:blue;
    font-weight:bold;
}
.col2 {
    background-color:#dddddd;
    text-align:center;
    font-family:Arial, Helvetica, sans-serif;
    font-size:12px;
    color:black;
    font-style:italic;
}
</style>

```

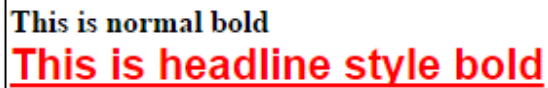
Selectors are key to styles. Once the selector is defined, they just need to be referenced in order to activate the style. One place styles can be defined is in the head of a document. In the following example a headline style is defined for within bold classes. The style can then be activated in the body of the document by referencing class="headline" as a parameter in the appropriate tag.

```

<html>
<head>
<style type="text/css">
    B.headline {
        color:red;
        font-size:22px;
        font-family:arial;
        text-decoration:underline
    }
</style>
</head>
<body>
<b>This is normal bold</b><br>
<b class="headline">This is headline style bold</b>
</body>
</html>

```

This body contains two lines styled with bold text, but the second line is also styled with the headline style. The webpage would appear as below with the second line, red, larger and underlined.



This is normal bold
This is headline style bold

Figure 2.2: A Simple CSS Style Example

2.3 Selectors

There are 3 types of selectors; HTML selectors, class selectors and id selectors. The selectors demonstrated in the introduction were all class selectors. HTML selectors are used to redefine existing HTML tags, as using css we can redefine the appearance of tags such as “”, “<P>” or “<H1>”. Class selectors are used to define new styles that can be used repeatedly whenever needed across a website. ID selectors are used for unique elements, when an element only occurs once on the page.

HTML selectors are used to redefine the existing HTML tags. The general syntax for HTML selectors is;

```
HTMLSelector { Property:Value; }
```

For example we can redefine the way the bold tag displays by adding the following, it should be clear what effect the styling will have.

```
<style type="text/css">
  B {
    font-family:arial; font-size:14px; color:red;
  }
</style>
```

Class selectors are used when a style may occur multiple times across a website, and we may need to call the style more than once. The general syntax for class selectors is;

```
.ClassSelectorName { Property: Value; }
```

For class selectors, you can choose what to name the class, but notice the dot before the class name. Class selectors can then be referenced within other tags in the body of the document. For example;

```
<html><head>
  <style type="text/css">
    .title {font-family:arial; font-size:14px; color:red}
  </style>
</head><body>
  <b class="title">This is a bold tag carrying the title class</b>
  <i class="title">This is an italics tag carrying the title class</i>
</body></html>
```

In this example the title styling is added to the bold and italic styling as follows.

This is a bold tag carrying the title class *This is an italics tag carrying the title class*

Figure 2.3: An Example Of A Class Selector

ID selectors are used to define elements on the page that are unique, where the style is only going to be used once. One place this can be useful is when creating layers, as normally each layer is a unique element. The general syntax for an ID selector is;

```
#IDSelectorName { Property: Value; }
```

Notice the hash before the selector name, this distinguishes the style from class styles. ID selectors can then also be referenced within other tags in the body of the document, as follows.

```
<html>
<head>
  <style type="text/css">
    #layer1 {position:absolute; left:100; top:100; z-Index:0; background-color:#FF9}
    #layer2 {position:absolute; left:120; top:120; z-Index:1; background-color:#6CC}
  </style>
</head>
<body>
  <div ID="layer1">
    THIS IS LAYER 1<br>POSITIONED AT 100,100
  </div>
  <div ID="layer2">
    THIS IS LAYER 2<br>POSITIONED AT 140,140
  </div>
</body>
</html>
```

In this example 2 layers are created and positioned on the webpage. In chapter 3 we will discuss further how CSS is used to manage the layout of the webpage, but for now notice the page would create 2 layers, with the 2nd layer positioned on top of the first layer, as shown below.

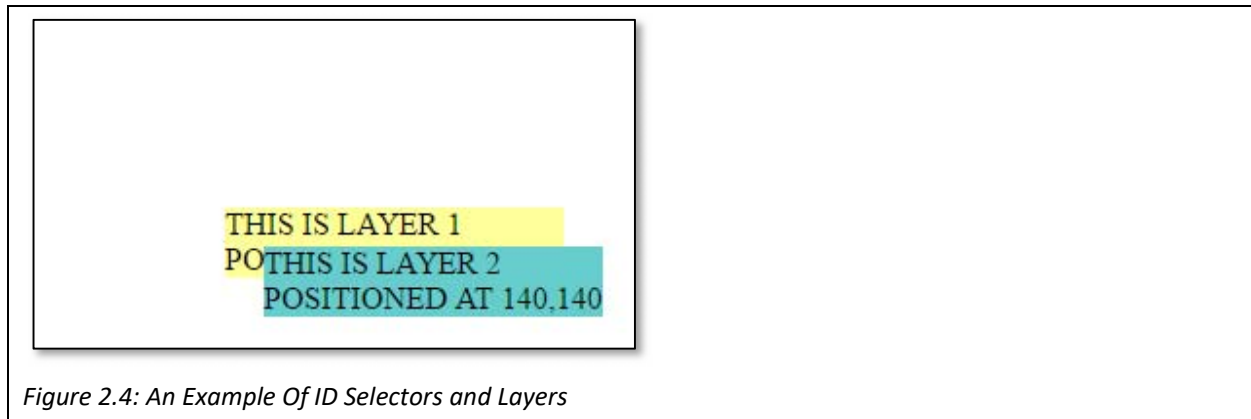


Figure 2.4: An Example Of ID Selectors and Layers

Quick Tip

You don't need to memorise all the different properties you can use style something, if you use a free program like "Topstyle Lite", it will give you a list of possible properties, and their corresponding values to choose from. The Dreamweaver development environment also offers the same feature.

2.4 & <Div>

In the example for ID selectors, the styles were referenced with a <div> tag. Div tags are particularly useful when used in combination with class and id selectors. Along with the tag, div tags are dummy tags, which don't really do anything themselves, but can be used to carry css styles. The span tag is an in-line tag, which means that no line-breaks are inserted before or after it. The div tag is a block tag, which means a line break is inserted around the div tags to separate it from the surrounding content (like <p> or <table> tags). The difference between span and div tags, unless other css is referenced, is the line break added by the div tag. Div therefore is particularly important for layers or other blocks of information. Take a look at the source of any modern webpage and you will find it broken down into blocks using div tags.

The difference between span and div tags, unless other css is referenced, is the line break added by the div tag

2.5 Where to CSS?

CSS can be added to a website at 3 levels. Styles can be added inline to a single, individual tag, or it can be added to the head of a page, as we have been doing so far. The preferred way to add css however is to separate it into a different file which can then be included on all pages maintaining a single style sheet across all the pages on the website.

Style can be added to any single tag by adding styles to the style parameter within any existing tag.

```
style = "styleproperty:stylevalue"
```

For example;

```
It is <b style="font-size:16px;">NOT</b> me.
```

In this example, the word "NOT" would be emphasised both with bold, and by changing the size of the font to 16 pixels. Single tag styling should be avoided wherever possible as using them loses out on many of the benefits of css. Needing to define styles for each and every tag means needing to define a style again and again, which is of course error prone and makes maintenance much more difficult, as changing a style means changing it in many places.

In the examples so far the style was defined in the page header, which makes that style available throughout the page. In this scenario the style can be defined once in the header and then used as many times as needed within the page. This allows more freedom with changing the style of a page after it has been made. As well as this being an advantage to the designer, it is also an advantage to the page visitor as pages will load more quickly as pages are smaller with styles only being defined once. This advantage is amplified when we consider using an external style sheet.

CSS can be defined for an entire site by simply placing the style definitions in a plain text file, and referring to that file in the header of each of the pages. This style sheet is then loaded the first time a visitor comes to your site and then cached for use when the user visits other pages on your site. This of course speeds up your sites performance, but also gives lots of flexibility for changing the style of a site after it has been made. In this scenario the style is completely separated from the content as no styles are defined on the page, they are simply referenced by a line in each pages header.

```
<link rel="stylesheet" href="mystyle.css" type="text/css">
```

Whenever this line is placed. It tells the browser to load the file called mystyle.css, located in the same folder as the html file and include its style definitions. CSS stands for cascading style sheets, and the word cascading is used to suggest which styles have the highest priority. If different styles are set in-line, and in an external style sheet, the browser needs to know which style to use. The highest priority style is in-line, defined within an HTML tag – this allows the designer to override any particular style if absolutely necessary. After in-line styles, the styles defined in style sheets are used, and if no style is applicable, the browser default is used.

2.6 More on selectors

In order to have a consistent website, it is common that multiple styles will share the same properties and values, for example using the same fonts, colors etc. If so selectors can be grouped, assigning certain properties to multiple selectors in one go. Consider the following selectors;

```
.heading { font-family:arial; color:red; background:blue; font-size:14pt; }  
.subheading { font-family:arial; color:red; background:blue; font-size:12pt; }  
.details { font-family:arial; color:red; background:blue; font-size:10pt; }
```

Here much of the style information for all three selectors is the same. A better way of defining this would be to group this information, defining their common properties together in one place.

```
.heading, .subheading, .details { font-family:arial; color:red; background:blue; }  
.heading { font-size:14pt; }  
.subheading { font-size:12pt; }  
.details { font-size:10pt; }
```

Clearly defining the styles in this way takes less space, is easier to maintain and is less error-prone. Notice that selectors that should share the same styling are separated by a comma.

A further consideration when defining selectors is to define context sensitive selectors, styles which should only be invoked in certain situations. For example we may want to define a style for the bold tag which is only invoked when the style is both bold and italic, but not when it is only bold.

```
<b><i>Style invoked here</i></b>  
<b>But not here</b>
```

This is possible by defining the following selector.

```
B I {font-size:16px; }
```

Here bold and italic html selectors are separated by a space, and not a comma. Grouped selectors and contextual selectors can be defined at the same time.

```
B I, .heading, B .subheading { font-size:16px; }
```

In this example the font-size 16px property would be active for text that is bold and italic, or with the heading class, or bold text with the subheading class. These examples demonstrate the flexibility of css selectors, a good front end developer will consider different ways to efficiently define their styles.

2.7 CSS Properties

There are many different properties that can be set using CSS. In this section we list some of the properties and introduce how their values can affect the appearance of various page elements, beginning with how colors are defined in CSS.

2.7.1 Color & CSS

Colors are composed of a combination of red, green and blue, with different colors being created by the amount of these colors added to the combination. Colors can then be specified in 3 different ways; using a color name, using a HEX value or using an RGB value. Firstly, there are 140 standard color names supported by all browsers, so colors can be set by simply specifying the name of the color.

```
color: red;
```

The 140 standard color names range from the common, like green, black and white, to colors such as “blanchedalmond” and “goldenrod”. It is worth noting that these color names are case insensitive, i.e. “crimson” is the same as “Crimson”.

Clearly if you want to design your webpages precisely, only 140 different colors is limiting, so the second way of specifying a color is using a HEX value. HEX stands for hexadecimal, or base 16, where the symbols (letters) A-F are used to represent the numbers from ten to fifteen, and sixteen is represented as 10. In this way the numbers between 0-255 are written as 00-FF, storing 256 different combinations using 2 characters (or a byte). Two characters are then used to store the amount of red in the specified color, two more for the amount of green, and finally two for the amount of blue. This gives a total of more than 16 million different combinations, represented by 6 characters. Some examples of these colors are given as follows;

color: #FF0000 – Red (the maximum value (FF) for red, and none (00) for blue and green).
 color: #00FF00 – Green
 color: #0000FF – Blue
 color: #FFFF00 – Yellow
 color: #000000 – Black
 color: #FFFFFF – White

Note that sometimes only 3 characters are used, with only one symbol for the amount of red, green and blue respectively, here red could be simply specified as #F00, but clearly this reduces the variety of different color combinations.

The third way of specifying a color is using RGB values. Here the amount of red, green and blue are specified in decimal. Here values between 0 and 255 are specified separated by commas, for example;

color: rgb(0,0,0);
 color: rgb(255,0,0);

There are many different colors to choose from, and many different color combinations, some of which are more visually appealing than others. There are several “color pickers” on the web that allow you to choose appealing color combinations and discover the appropriate HEX or RGB values.

2.7.2 Text & CSS

There are many properties that can be used for styling text, including;

Property	Values	Notes
color	Any color name, HEX or RGB value	To be W3C compliant, if you set a font's color, you should also define the background-color property
text-align	left, center, right, justify	left is the default
text-decoration	none, overline, line-through, underline	Setting text-decoration to none is a useful way of removing the default underline from links
text-transform	uppercase, lowercase, capitalize	
text-indent	Value in pixels	Indents the first line of text
direction	rtl	rtl stands for right to left
letter-spacing	Value in pixels	
word-spacing	Value in pixels	
line-height	Value in pixels	

Table 2.1 Text Styling Properties

Related to the text CSS properties, there are also several properties that can be used related to the font. The first of these to investigate is the font-family property, which is used to specify which font should be used to display the text. Generally the font-family property will contain a list of different fonts, beginning with the preferred font, and ending with a generic font to be used if the desired font isn't available. Note that if the font name contains spaces it must be surrounded by quotes. Some fonts may not be available on different computers, or in certain browsers, indeed maintaining consistency across multiple browsers is a big challenge that web designers face continually. If necessary, a specific font can be sent to the browser to be used, but generally a list of options will suffice.

```
p { font-family: "Times New Roman", Georgia, Serif; }
```

The size of the font is set using the font-size property. Just as in any word processor, this can be set to a specific number of pixels, for example;

```
H1 { font-size: 40px; }
```

Setting the font-size by specifying the number of pixels is an absolute setting, which is an appealing choice as it allows the designer to precisely manage the look of the site – to pixel precision. However, many web users, for instance visually impaired users, like to control their browsing experience, and most browsers support the ability to increase or decrease the font size. Setting the font-size in pixels doesn't scale up for users who want to zoom the text, or down for viewing on mobile devices. Alternatives to using pixels to specify font size include percent and Ems, which are relative settings.

An "Em" originally referred to the width of the letter "M", but in typography refers to the size of a letter relative to the currently specified size. 1em is equivalent to the current font size, which by default in browsers is 16px. However, by setting the font-size using em, rather than px, it allows users to adjust the size they view, as the true size is dependent on zoom settings and the DPI setting for the screen. Assuming the current font-size is 16px, then setting the font-size to 1em would keep the font-size at 16 pixels, while a setting of 2.5em would make the font-size 40px (2.5 * 16). Alternatively a setting of 0.5em would result in font-size 8px (0.5*16).

```
H1 { font-size:2.5em; /* 2.5*16 = 40px */ }  
.smalltext { font-size:0.5em; /* 0.5*16 = 8px */ }
```

Note that these examples can cause problems in older versions of internet explorer, and you have no control over which browser your visitors use. A further alternative is to use percentage, which has a similar relative text size effect as when using em.

Two further text related properties are font-weight and font-variant. For font-weight, the options are normal or bold, offering an alternative way of creating bold text. For font-variant the options are normal or small-caps, where lowercase characters are made uppercase, but in a smaller font size.

2.7.3 Backgrounds & CSS

CSS makes it easy to design the background of an element or a page. While this section gives examples of setting background properties for the whole body of a page, the same principle can be applied to any

element on the page. At its simplest, the background color of an element can be set, by using the background-color property and using one of the coloring methods discussed previously.

```
body { background-color: #ff0000; }
```

Alternatively, an image can be used as the background of an element, and for this the url of the image must be specified, in a similar way to when placing an image on the page. In this case the background-image property is set.

```
body { background-image: url("background.jpg"); }
```

Care should be taken when selecting background images, as often text will be placed on top of the background. Choosing the wrong background can make text unreadable, bearing in mind you may not know the width and height of your visitor's browser window. By default the background image specified will 'repeat' itself, both horizontally and vertically, which naturally may not be the intention. This can be controlled by the background-repeat property, which can be set to "repeat-x", "repeat-y" or "no-repeat" to specify a horizontal, vertical, or no repeat respectively.

While we will discuss positioning elements further in the next chapter, it may be that the background image needs to be positioned, not in the top left corner of the element – particularly if the image is not to be repeated. The background image can be placed using the background-position property, which can take combinations of the following values; "top", "center", "bottom", "left" and "right". By default the background image will scroll with the element, which can be stopped by setting the background-attachment property to be fixed. All of these background settings could lead to CSS as follows;

```
body {  
    background-color: #ffffff;  
    background-image: url("background.jpg");  
    background-repeat: no-repeat;  
    background-position: top center;  
    background-attachment: fixed;  
}
```

As this is somewhat longwinded, it is possible to set all these properties in a single statement, using the shorthand method;

```
body { #ffffff url("background.jpg") no-repeat fixed top center; }
```

This works so long as the properties are specified in the correct order; background-color, background-image, background-repeat, background-attachment, background-position. However, it doesn't matter if some properties are missing.

2.7.4 Links & CSS

The traditional HTML link appears in blue, underlined, but unless the website is designed for the 1990s, links can be designed so much more attractively. Crucially a link needs to stand out from the rest of the page, so that users know they should click on it, but there are different ways that designers can make links stand out.

Obviously when styling a link, CSS is added to the anchor tag, or “A” tag. A common example is to remove the underlining, which is done by setting the text-decoration parameter for the anchor tag;

```
a:link {text-decoration: none; }
```

Notice the syntax here for “a:link”, the colon is used to define a style for a particular pseudo-class. A pseudo-class is used to define a particular state for an element, in this case the style is used when the link is in its link state. A link can however be in other states and initiate an alternative pseudo-class; namely a link which has already been visited, a link which is being hovered over by the mouse, and a link which has been clicked – pseudo-classes can be used to define different styles for each of these scenarios.

```
a:link { color: #000000; }  
a:visited { color: #404040; }  
a:hover { color: #8c8c8c; }  
a:active { color: #bfbfbf; }
```

Using this CSS will make the links a different shade of gray (to black) depending on their state. Note that when the mouse hovers over a link this will change its state, and so change its color, similarly with the active state when the mouse clicks on the link. Using CSS to change properties like this can begin to activate a website, and in the next chapter we will see an example of a CSS controlled drop down menu when a mouse hovers over a menu item.

2.7.5 Lists & CSS

Chapter 1 briefly introduced lists in HTML. HTML supports two different kinds of lists, ordered and unordered. An unordered list, as seen in chapter 1, would appear with bullet points, and is indicated by the tag. An ordered list could appear as a numbered list, and is indicated by the tag. In each case each list item is indicated using the tag. CSS allows the designer to style these lists in more interesting ways, mostly using the list-style-type property. For unordered lists, the default marker is a disc, but the alternatives are a circle, or a square;

```
ul { list-style-type: square; }
```

Alternatively a custom image could be used by specifying the list-style-image property;

```
ul { list-style-image: url("my_marker.gif"); }
```

For ordered lists there are a variety of options depending on whether the list should be ordered by numbers or letters. Simple parameters for the property include Armenian, Georgian and Katakana numbering systems, but more useful values include “lower-alpha”, “upper-alpha”, for lower and upper case alphabet. The “decimal” value is used for standard numbering, while roman numerals can be easily used using “lower-roman” or “upper-roman”.

```
ol { list-style-type: lower-alpha; }
```

Key Points

- CSS is used to control the style of a website; all aspects of the presentation including layout, colors and fonts.
- Existing HTML selectors can be redefined to make them appear differently, while new styles can be created using either class or ID selectors.
- ID selectors should be used when there is an element that will only occur once, while class selectors are used for styles that occur multiple times.
- Class selectors are defined using a dot ".", while ID selectors are defined using a hash "#".
- and <Div> are dummy tags that can be used to invoke style, while not really doing anything themselves – the difference is that the <Div> tag will insert a line break around it.
- CSS can be defined either within a single tag, within the head of a page, or in a separate stylesheet.

Further Resources

- 1) The w3schools website has a tutorial introducing the basics of using cascading stylesheets – their site can be found here:-
<https://www.w3schools.com/css/>
- 2) Html.com has a guide for using css alongside html. Their guide can be found here:-
<http://html.com/css/>
- 3) Csstricks.com has a wide variety of ideas for how CSS can be used to improve the design of websites. Their site can be found here:-
<https://css-tricks.com/>

Assignment

You should now be able to get going with CSS! Dive in! Take the simple web page created at the end of chapter 1 and 'pimp it up'. Use all the CSS techniques described in this chapter to design a page that looks modern and stylish.

Chapter 3

CSS for Layout

Objectives

This chapter continues the exploration of Cascading Style Sheets, this time investigating how it can be used to manage the layout of a page. The layout of a page is important to make sure each element on the page appears where it is meant to be, particularly in today's world where website visitors are using different browsers on a variety of devices with different screen sizes. Making pages *responsive* to being opened on different screens is an important goal for a web designer. After reading this chapter you should:

- Understand the different settings that can be applied to the position property, notably the fixed, relative and absolute properties, and how they can be combined to position elements inside other elements.
- Understand how elements can be floated next to each other to create multiple columns, managing situations where different columns have different heights.
- Be comfortable with the box model, knowing how each element is surrounded by padding, borders and margins and being able to control an elements position by adjusting these values.

Contents

- 3.1 The Position Property
- 3.2 Float and Clear
- 3.3 Borders, Margins and Padding
- 3.4 A Simple CSS Image Gallery
- 3.5 A Simple CSS Hoverbox

Chapter two introduced how CSS is used to manage the style of a webpage, this chapter explores how CSS manages the layout of a webpage. Generally a webpage will consist of many elements, or blocks. Some elements will contain content like text, others may contain images, and others may contain navigation tools, while some may contain adverts. An important task for a web designer is to make sure all the elements appear where they are meant to appear, in all browsers, with potentially different devices and screen sizes. Fortunately CSS provides all the necessary tools for managing a webpage's layout.

The previous chapter introduced the and <div> tags, the <div> tag in particular is very useful for arranging a page into blocks and ensuring they appear where they are intended to. The previous chapter also gave a very brief example that introduced positioning layers on a webpage. Let's revisit that example;

```
<html>
  <head>
    <style type="text/css">
      #layer1 {position:absolute; left:100; top:100; z-Index:0; background-color:#FF9}
      #layer2 {position:absolute; left:120; top:120; z-Index:1; background-color:#6CC}
    </style>
  </head>
  <body>
    <div ID="layer1">
      THIS IS LAYER 1<br>POSITIONED AT 100,100
    </div>
    <div ID="layer2">
      THIS IS LAYER 2<br>POSITIONED AT 140,140
    </div>
  </body>
</html>
```

In this example, two ID selectors are created and for each ID, 5 CSS properties are set. HTML then creates two layers, one for each of the ID selectors. From chapter two the background-color property should be easily understood. The other 4 properties are the interesting part when using CSS for layout. The left & top properties should be straightforward, they set how many pixels from the left and top of the screen that each block should appear. The z-index property sets which element should appear on top of another element, with elements that have a higher z-index property appearing on top. Finally the position property is perhaps the most interesting, in this case the position is set to absolute, which means that the elements are removed from the page, and positioned absolutely as specified. The result is two coloured blocks that are positioned away from the top left corner, with one block appearing over the other block.

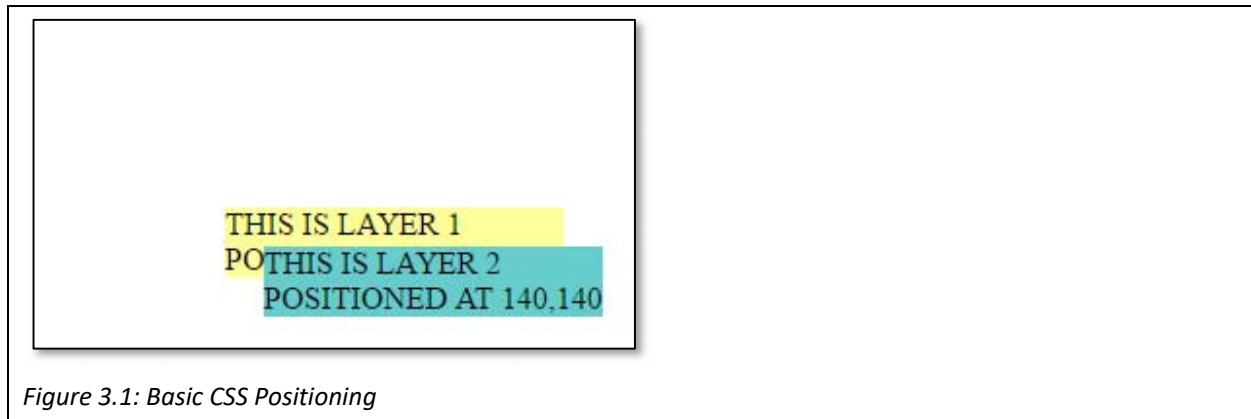


Figure 3.1: Basic CSS Positioning

3.1 The Position Property

The position property can be set for any selector, and has 4 important possible values; static, absolute, relative and fixed. Firstly, the static value is the default value, i.e. the way all our examples have worked so far. The *static* value means that the element is not affected by properties such as top, bottom, left or right, indeed the element is not positioned in any special way, it is just positioned in the normal flow of the page, following the elements positioned before it. The CSS is straightforward, the following would make any div with the class static to be positioned statically. Note that this would not be necessary as divs are positioned statically by default.

```
div.static { position:static; }
```

The second positioning value is the *relative* value. The relative value allows the element to be moved away from its regular position by setting top, bottom, left or right properties. The element is then positioned relative to its normal position. The content around it will not be adjusted though, so the relative value could leave gaps on the page, or move elements to appear over other elements. In the following example a div with the relative class would be moved 20 pixels down from the top and 20 pixels away from the left.

```
div.relative { position:relative; left: 20px; top: 20px; }
```

We can see these effects in a fuller example;

```
<html>
<head><style type="text/css">
  div.static { position:static; }
  div.relative { position:relative; left: 20px; top: 20px; }
</style></head>
<body>
<div class="static">This div is statically positioned!</div>
<div class="relative">This div is relatively positioned!</div>
</body>
</html>
```

This HTML will produce a page that looks as follows;



Figure 3.2: Static And Relative Positioning I

As can be seen, the second div has been moved 20 pixels to the right and 20 pixels down from where it would normally follow the first div. Notice what happens if we switch the two divs around, and place the relatively positioned div on the page before the statically positioned one. In this case the relatively positioned div is moved on top of the statically positioned one. This is because it has been removed from the regular flow of the page. Notice the white space that has been left in the top left.

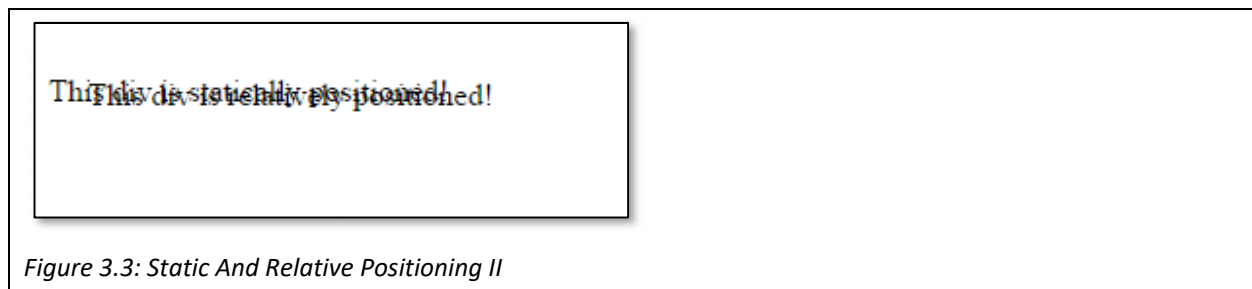


Figure 3.3: Static And Relative Positioning II

The *fixed* position property places an element in a fixed position within the viewable area. In this case the element will remain fixed even when the page scrolls. The position is set using the top, left, right or bottom properties, so for example if an element was needed to be fixed in the top right corner of the page, it could use the following class. Note that the width property has also be set to limit the size of the element.

```
div.fixed {  
    position: fixed;  
    top: 0px;  
    right: 0px;  
    width: 200px;  
}
```

The 4th important position value is *absolute*, as seen in the opening example. Like a fixed element, absolutely positioned elements can be precisely placed on the document, however, unlike fixed elements, the absolutely positioned element will scroll with the page. An absolute element is positioned relative to its nearest ancestor, which means that when an absolutely positioned element is placed inside another element the element is positioned in relation to the containing element. In the opening example, neither layer 1 or layer 2 were inside another positioned element, so they were placed in relation to the document's body. Some useful

positioning effects can be gained from absolutely positioning elements inside other elements, such as inside a relatively positioned element, as can be seen in the next example.

```
<html>
<head>
<style type="text/css">
  div.relative { position:relative; width:400px; height:200px; border: 1px solid black; }
  div.absolute { position:absolute; width:200px; height:100px; top:80px; right:20px; border: 2px solid black; }
</style>
</head>
<body>
  <div class="relative">
    This div is relatively positioned!
    <div class="absolute">
      This div is absolutely positioned!
    </div>
  </div>
</body>
</html>
```

Here there are 2 divs, one relatively positioned and one absolutely. The relatively positioned div is given a size of 400 pixels by 200 pixels, while the absolutely positioned div is smaller at 200 pixels by 100 pixels. The relatively positioned div has a thin black border, while the absolutely positioned one is thicker. Notice that the absolute div is nested inside the relative div, which means that it is to be positioned in relation to it – in this case the relative div is the parent of the absolute div. We will explore the idea of parent and child elements later when we introduce the Document Object Model, or DOM. For now, the absolute div will be placed 80 pixels from the top and 20 pixels from the right, inside the relative div, as follows.

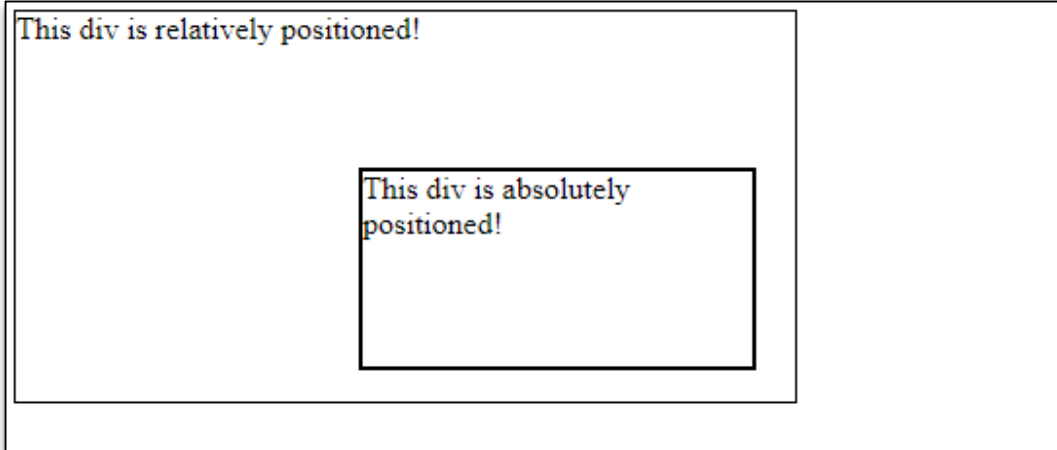


Figure 3.4: Relative And Absolute Positioning

Using this technique a webpage can be broken down into blocks, and sub-blocks, with the elements of each sub-block being positioned within the block. Different blocks may contain images, text content, navigation etc. CSS can also be used to create tables, or column layouts by simply placing multiple absolutely positioned divs inside a relatively positioned div. These four positioning values are important for managing the layout of a page using CSS.

Property	Effects
static	Default setting, element will follow previous element
relative	Element is moved relative to itself, i.e. moved from where it would statically appear, based on the top, left, right and bottom settings
absolute	Element is moved to an absolute position within its parent element
fixed	Element is fixed in position in the viewport, and will not move as the page scrolls

Table 3.1: The Four Positioning Properties And Their Effects

3.2 Float & Clear

Using relatively and absolutely positioned elements it is straightforward to create multiple columns. Consider this 2 column example using dummy text. The first column is defined with the class col1, positioned absolutely on the left, while the second column is defined with class col2, positioned absolutely on the right. As both columns are placed within a relative block, they appear next to each other.

```
<html>
<head>
<style type="text/css">
  .relative { position:relative; width:500px;}
  .col1 { position:absolute; top:0; left:0; width:250px;}
  .col2 { position:absolute; top:0; right:0; width:250px;}
</style>
</head>
<body>
  <div class="relative">
    <div class="col1">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore
    </div>
    <div class="col2">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
      dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip
      ex ea commodo consequat.
    </div>
  </div>
  This content will cause a problem.
</body>
</html>
```

A big problem with this example occurs if we try to add any further content after the relative div, before the closing body tag. Even though the relative div contains two further divs, because they are absolute divs, they have been removed from the general flow of the document to be positioned absolutely. This means the relative div has no height, and so any content following this div will appear in the same place as can be seen below. One simple solution could be to simply add a height parameter to the relative div, however often the amount of content to be included in columns is unpredictable, and can be affected by things such as the font size. In this example, each column contains a different amount of content, and sometimes the content may be images rather than text.

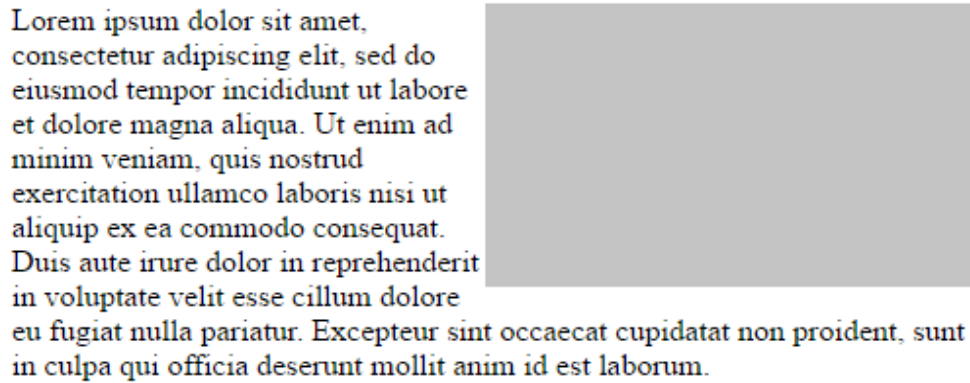
The important bit on this example. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Figure 3.5: Problem With Relative Positioning

If the columns may have varying heights, then absolute positioning won't work. In this scenario, an alternative solution is to float elements, so that they are pushed as far to the left, or right as they can. The typical use for this is to wrap text around an image, but it can also be used for more complex layout tasks. The float property can be set to float elements to the left or right.

```
<html><head>
<style type="text/css">
  .relative { position:relative; width:500px;}
  .float { float:right;}
</style>
</head>
<body>
  <div class="relative">
    <div class="float">
      
    </div>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
    dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex
    ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu
    fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
    mollit anim id est laborum.
  </div>
</body></html>
```

In this example, text is wrapped neatly around a simple grey image which has been floated to the right. Because the text is now part of the relative div, any content that follows it will appear as expected.



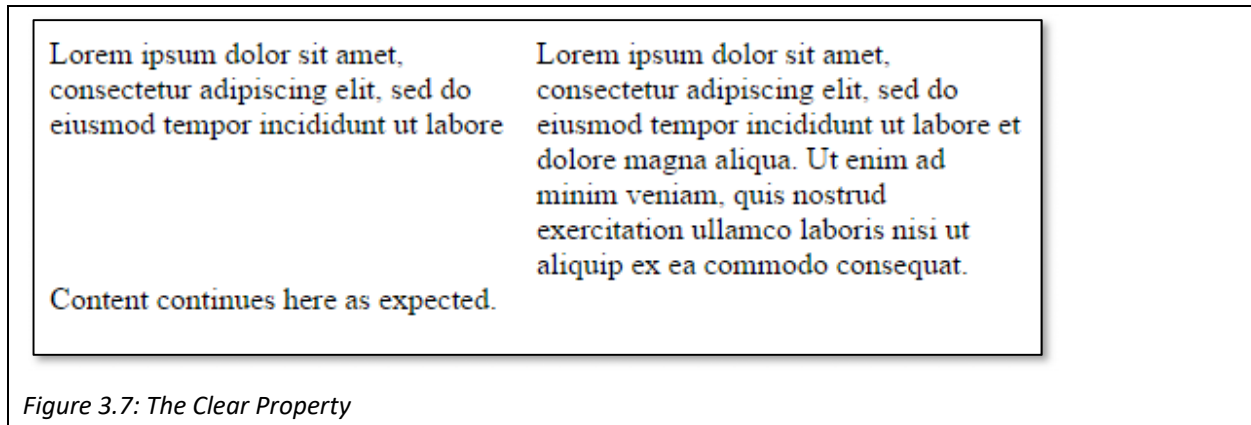
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figure 3.6: An Example of Float

Rather than wrapping text around an image, the float property can be used to float multiple columns next to each other, with varying column heights. When an element, such as the grey picture above, is set to float, then subsequent content will flow around it. This would mean that all content that follows a floated section will flow around it. To end the floating section, the clear property can be applied.

```
<html><head>
<style type="text/css">
  .relative { position:relative; width:500px;}
  .col { float:left; width:250px;}
  .clear { clear:both; }
</style>
</head>
<body>
  <div class="relative">
    <div class="col">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore
    </div>
    <div class="col">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
      dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip
      ex ea commodo consequat.
    </div>
  </div>
  <div class="clear"></div>
  Content continues here as expected.
</body></html>
```

The clear property can have several values; left, right or both. Left and right can be used to only clear certain floats, while both will clear all floats. In the example given, the clear selector has been created to clear both sides, and is then carried in an empty div following the columns. The result successfully creates two columns of different heights, and potentially different widths, with the following content appearing afterwards as intended.

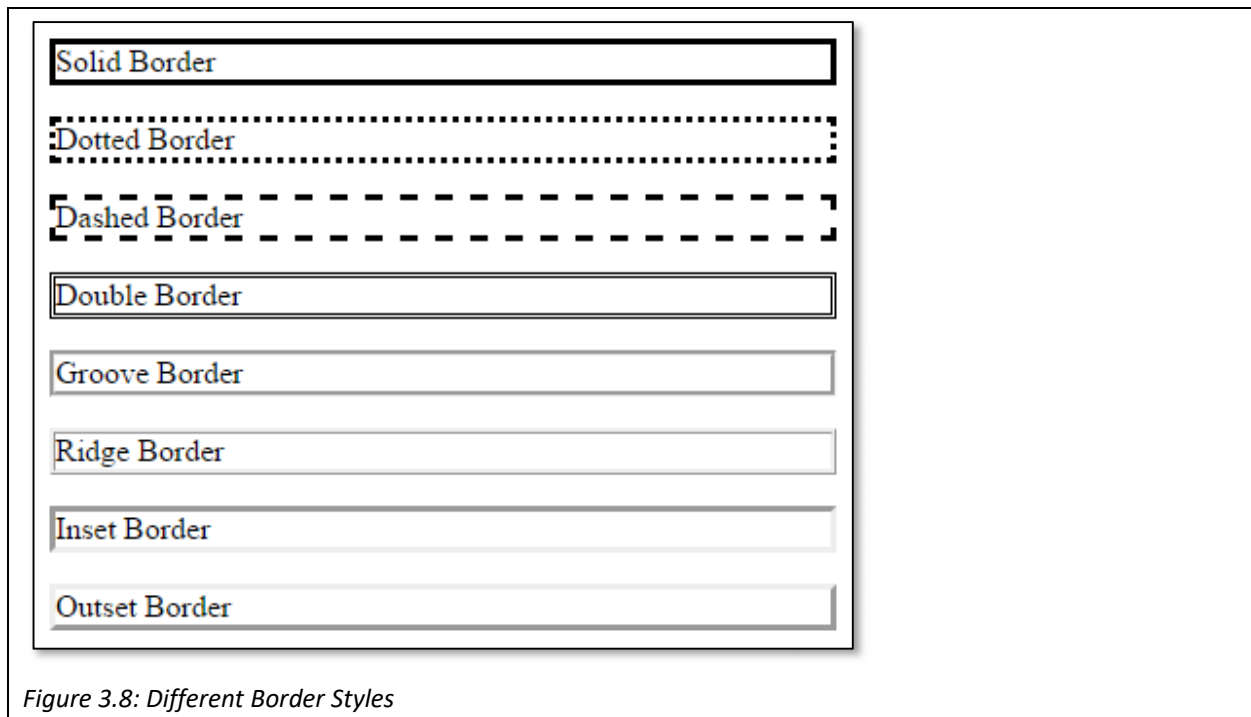


3.3 Borders, Margins & Padding

In a previous section we demonstrated how to set a simple border around an element;

```
.simpleborder { border: 1px solid black; }
```

This example demonstrates the shorthand for setting border properties. In this one line 3 border related properties are set; the width, the style and the color.



The width property should be self-explanatory, simply specified in pixels, similarly the color property is set using standard CSS colors. There are several options though for the border-style property. The possible values are; solid, dotted, dashed, double, groove, ridge, inset and outset, which provide the following effects. The border-style property must be set, i.e. a solid border is not a default, without a border-style, the other border properties will not take effect. The width can be set precisely using pixels, or as thin, medium or thick. The default border-width, as used in the previous graphic, is medium. While a medium border appears to be 4 pixels, this may depend on the browser specification, so for precision it is worth using a precise pixel value. It is also possible to mix the border and set different borders for each of the 4 sides of the element. The following are just examples of all the border related properties; border-bottom-color, border-left-style, border-top-size.

Any of the elements of a webpage could have a border around it, and largely the border is a stylistic aspect, so perhaps a topic for the previous chapter where CSS is used for style. However, borders also add size to an element, which can have a big impact on the layout of the page. In a previous example there were two elements with width 250 pixels that fitted neatly into an element of width 500 pixels. If these elements were to have borders, then they would not fit as the border adds pixels to the width. Outside of the border, each element also has a margin, which can add width (and height) to each element, while inside the border there is a padding, which can be used to space elements attractively.

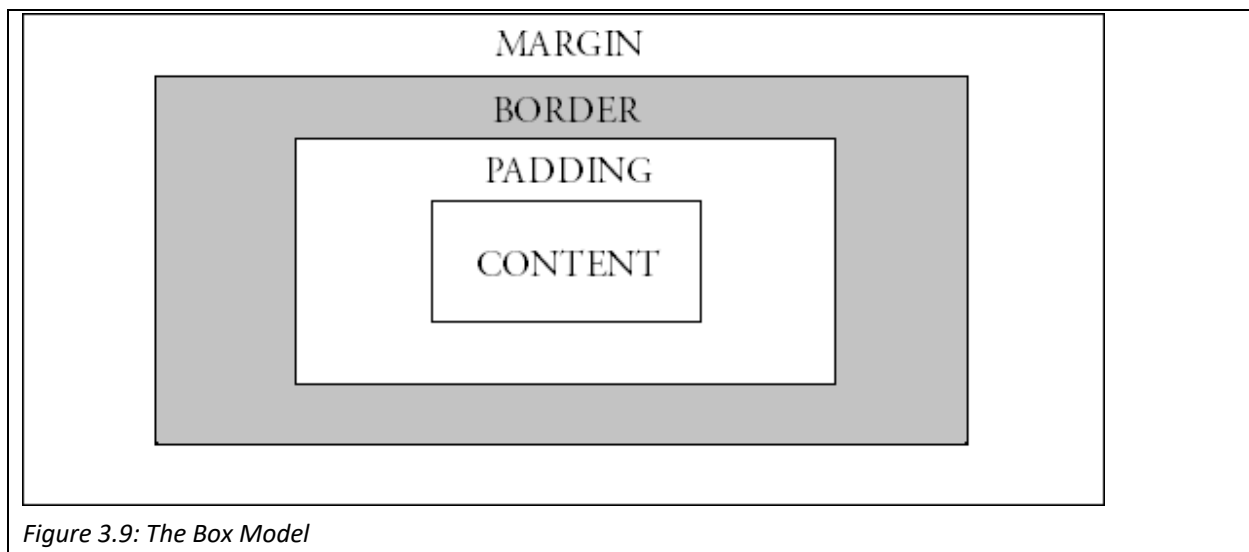


Figure 3.9: The Box Model

Often referred to as the Box Model, each element can be considered as a box, which is surrounded by padding, and then a border, and then a margin. The margins can be set using the margin-top, margin-right, margin-bottom, margin-left properties. Similarly the padding can be set using the padding-top, padding-right, padding bottom and padding-left properties. In both cases it is common practice to use the shorthand version and set all 4 properties in one line, the following 2 examples are equivalent;

```
.margins { margin-top: 5px;  
margin-right: 4px;  
margin-bottom: 3px;  
margin-left: 2px; }
```

```
.margins { margin: 5px 4px 3px 2px; }
```

If the margin (or padding) property has 4 values, such as this, then they are applied to the top, right, bottom and left edges in that order. If the margin (or padding) property has 3 values, then the first is applied to the top, the second is applied to the left and the right edge, while the third is applied to the bottom. If the margin (or padding) property has 2 values, then the first value is applied to the top and bottom and the second applied to the left and right, while if there is only one value, it is applied to all 4 sides, for example;

```
.margins { margin: 1px; }
```

On further value that the margin property can take is the “auto” value. This can be very useful for centering elements inside a larger element. In the following example, the left and right margins of the element are set to auto. Here, assuming the width of the element has been set, then the remaining space either side is automatically split between the left and right margins, hence centering the element. This is particularly useful when centering the whole page block within the body tags – while we don’t know the size of the user’s screen, we can use the auto value to position the page in the center of the viewing area, rather than in the top left corner, no matter how big the screen is. This works nicely, assuming the browser window is wider than the element’s width, otherwise the browser may add a horizontal scrollbar. An improvement can be to change the width property to use the max-width property, particularly when designing for mobile environments.

```
.margin { width: 600px; margin 0 auto; }
```

An issue here which has caused a mathematical headache for web designers is that the true width of an element is the width + padding-left + padding-right + border-left + border-right + margin-left + margin-right. Similarly the true height of the element needs to add the top and bottom padding, border and margin to the height of the element.

Quick Tip

If you are struggling to position an element precisely, try opening your page in the chrome browser. Right click on the element you are trying to position and select “Inspect”. The developer tools window will appear, and you can see your HTML and also the CSS that is being applied to the element. Here you can temporarily edit the CSS to find the values you really need.

There are alternative methods emerging, but as with many web development techniques, they may not work with all browsers, particularly older versions of a browser. So, another way of dealing with this issue is to set the width of an element using percentages, rather than pixels. If you want 2 columns to appear in a block, you can simply set the width of each column to be 50%, rather than calculating the number of pixels. Similarly if you want 3 columns, you could set the width of each to be 33%. Setting widths as percentages, otherwise known as using relative units, makes your pages more responsive, and better able to cope with different screen sizes and resolutions.

```
.splitDisplay{float:none;clear:left;}
.split50{width:47%;float:left;}
.split33{width:31%;margin-right:1%;float:left;}
.split66{width:64%;margin-right:5px;float:left;}
```

3.4 A Simple CSS Image Gallery

Later chapters will introduce javascript, which will enable you to create more sophisticated image galleries, but a simple image gallery can be created in simple CSS.

```
<html>
<head>
<style type="text/css">
    #content {max-width:800px; margin: 0 auto;}
    .image {float:left; margin:4px; border: 3px solid #ddd; padding:3px; }
    img { width:180px;}
    .image:hover {border: 3px solid #333;}
    .description {padding:5px; text-align:center;}
</style>
</head>
<body>
    <div id="content">
        <div class="image">
            <a href="black.jpg"></a>
            <div class="description">Black Image</div>
        </div>
        <div class="image">
            <a href="white.jpg"></a>
            <div class="description">White Image</div>
        </div>
        <div class="image">
            <a href="dark.jpg"></a>
            <div class="description">Dark Image</div>
        </div>
        <div class="image">
            <a href="light.jpg"></a>
            <div class="description">Light Image</div>
        </div>
    </div>
</body>
</html>
```


In this simple example, four images are floated next to each other, while the images could be anything, these images are simple colour spaces. The images appear inside a content block which has its width specified, but the content will appear in the middle. CSS is used to set the margins, borders and padding for each image, and these can be easily adjusted to the image gallery needs. Each image is contained within a link to the full version of the image, and each image is accompanied by a description which appears centered below the image. The image gallery is activated by using the hover pseudo-class. When the mouse is hovered over any of the images it is highlighted by making the border color is made darker.



Figure 3.10: A Simple CSS Image Gallery

3.5 A Simple CSS Hover Box

In future chapters we will investigate how CSS can work with javascript to activate webpages improving features like image galleries. In this section a simple information box is displayed when the user hovers their mouse over a button.

```
<html>
<head>
<style type="text/css">
  #content {max-width:800px; margin: 0 auto;}
  .button { border:2px solid black; width:200px; text-align:center; padding:3px; }
  .hoverbox {
    display:none;
    border: 1px solid #666;
    position:absolute;
    top:40;
    padding:3px;
    background-color: #ddd;
  }
  .button:hover .hoverbox { display:block; }
</style>
</head>
```

```

<body>
  <div id="content">
    <div class="button">
      Hover Here
    <div class="hoverbox">More information can be found here</div>
  </div>
</div>
</body>
</html>

```

In this example a simple button is created and styled. Then a hoverbox style is created with the display property set to "none". As expected this instructs the browser not to display the hover box. However, in the button's hover pseudo-class the display is set to "block", making it appear when the button is hovered over.

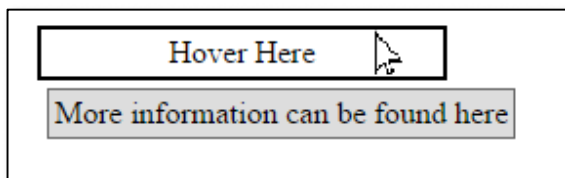


Figure 3.11: A Simple CSS Hover Box

Key Points

- CSS is also used to precisely position any element on a page so that it appears exactly where it is intended to appear on any device, responsively using any screen size.
- The position property can be set to be static, relative, absolute or fixed.
- Relative positioning moves an element relative to its original location in the document.
- Absolute positioning removes an element from the page and positions it precisely within its parent element.
- Fixed positioning allows an element to remain where it is even if the page is scrolled.
- Elements can be floated next to each other to create column layouts.
- The box model demonstrates how each element is surrounded by padding, a border and a margin – adjusting these properties will change how elements are laid out.

Further Resources

- 1) The w3schools website has a tutorial discussing CSS for layouts, which can be found here:-
https://www.w3schools.com/html/html_layout.asp
- 2) Learnlayout has an easy to read guide to the fundamentals of CSS for layout, which can be found here:-
<http://learnlayout.com/>
- 3) The Mozilla Developer Network has a detailed reference guide, complete with examples that can be found here:-
https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout

- 4) Bootstrap is an open source toolkit / framework for building responsive pages with HTML and CSS (and JavaScript). Once you are comfortable with the basics of creating CSS layouts, it is worth spending some time to use bootstrap help with your page layout. You can download Bootstrap from here:-
<http://getbootstrap.com/>

And, w3schools has a guide to using Bootstrap here:-

<https://www.w3schools.com/bootstrap/default.asp>

Assignment

The best way to learn how to layout a page is to jump in and try to recreate an existing page, with pixel precision! Consider the following 9 layouts – try to recreate them exactly, making sure they open correctly on different size screens.



Part 2

Server-Side Scripting with PHP

Chapter 4

Introducing PHP

Chapter 6

PHP Files & MySQL Databases

Chapter 5

PHP Functions and Objects

Chapter 7

Cookies, Sessions & Security

Part two focuses on the PHP programming language (or Hypertext Pre-Processor), a popular programming language used by as much as 80% of websites. The language is a core component of the WAMP (or LAMP / MAMP) stack, along with an operating system, Apache and MySQL. This part will explore how PHP is used on a server to generate pages that are then passed to the client machine to be displayed in a browser. Being run on the server means the code works closely with the server's filesystem and a database, such as MySQL. This part also deals with some security threats as well as managing cookies and sessions.

Chapter 4

Introducing PHP

Objectives

This chapter introduces the PHP programming language, an important server side language. Because PHP code is executed on a server, away from the browser, either a server is needed or a webserver can be set up on any machine. This chapter covers the basics of PHP, introducing how code can be run on a server to generate a HTML file to send to the client machine. After reading it you should;

- Be able to set up a webserver.
- Understand the basic structure of PHP code, noting the similarities and differences between PHP and other languages.
- Know how to use Variables in PHP.
- Be able to use operators in PHP.
- Understand the basic flow control statements used in PHP.
- Be able to create a simple form to pass data from one page to another.

Contents

- 4.1 Setting Up a Web Server
- 4.2 PHP Basic Syntax
- 4.3 PHP Variables
- 4.4 PHP Operators
- 4.5 PHP Flow Control
- 4.6 PHP Form Validation Example

PHP is a language where code is executed on the webserver, often before the page is sent to the user. It is an important language, currently used as the server side language for more than 80% of websites. Later in the book we will consider alternatives, but for now PHP is a core part of the WAMP development stack. It is important for web developers to understand the differences between running code on a server and running it on the user's machine. While a web developer can't decide which browser, and which browser version, the user is going to use, the developer has much more control over the web server. Running code on the server also gives easy access to databases stored on the server. Understanding when and where to run code is an important part of becoming a web developer.

PHP is an acronym for "Hypertext Pre-Processor", used to generate html code to be sent to the user's computer. PHP is a free, open source, server side scripting language, which is both powerful, and easy to learn.

PHP originally stood for Personal Home Page, but now it is an acronym for "Hypertext Pre-Processor", as it is used to generate html code to send to the user's computer. PHP is a free, open source, server side scripting language, which is both powerful, and easy to learn. PHP files have the extension .php and can contain both HTML and PHP code. When the webserver receives a request for a php file the request is sent to the php engine before HTML is returned to the machine where the request came from.

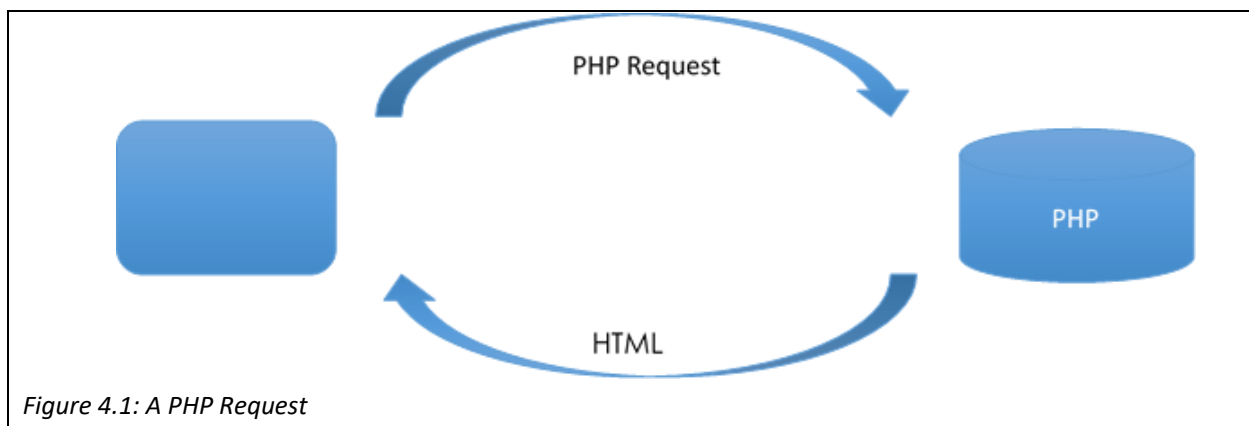


Figure 4.1: A PHP Request

4.1 Setting Up A Webserver

If you have access to a webserver online, it is likely to already have software running on it, and that is likely to include php amongst others. If you don't have a webserver already then it is still possible to set up your personal machine to create a development environment. To set up your machine, you will need to install a software stack, depending on your machine. In this book the WAMP software stack is used, and a version of it can be freely downloaded from wampserver.com. There are various alternatives such as LAMP, or MAMP, depending on your machine. WAMP stands for Windows, Apache, MySQL, PHP. These 4 software components are the 4 layers of the webserver that we will use.

The “W” clearly refers to the operating system, which in this environment is Windows. With a layered model, any of the layers can be replaced without affecting those around it, so Linux users can easily use LAMP instead. The “A” stands for Apache HTTP Server, the most widely used webserver software, which is free and open source, this is the core software which deals with HTTP requests. The “M” stands for MySQL, which is the database software that we will investigate in more detail in coming chapters, and the “P” stands for PHP. The WAMP software stack can be installed on any windows machine, to make it work like a webserver – from now on, rather than opening html files directly from the browser, they can be opened through Apache, using the localhost.

After installing webserver software, there will be a “www” directory, in which files can be saved, and then opened via localhost. To test the software is working, save a file called “hello.php” with the following contents in the “www” folder.

```
<?php echo “Hello World”; ?>
```

Next open this file in your browser at the url:- localhost/hello.php, or 127.0.0.1/hello.php. localhost is a hostname that refers to this computer, and generally resolves to the ip address 127.0.0.1. Assuming your webserver is running, you should see the following in your browser window, and now have a web development environment set up on your local machine. Remember, even though PHP code is running on your local machine, it is acting as a webserver that could be running anywhere in the world.

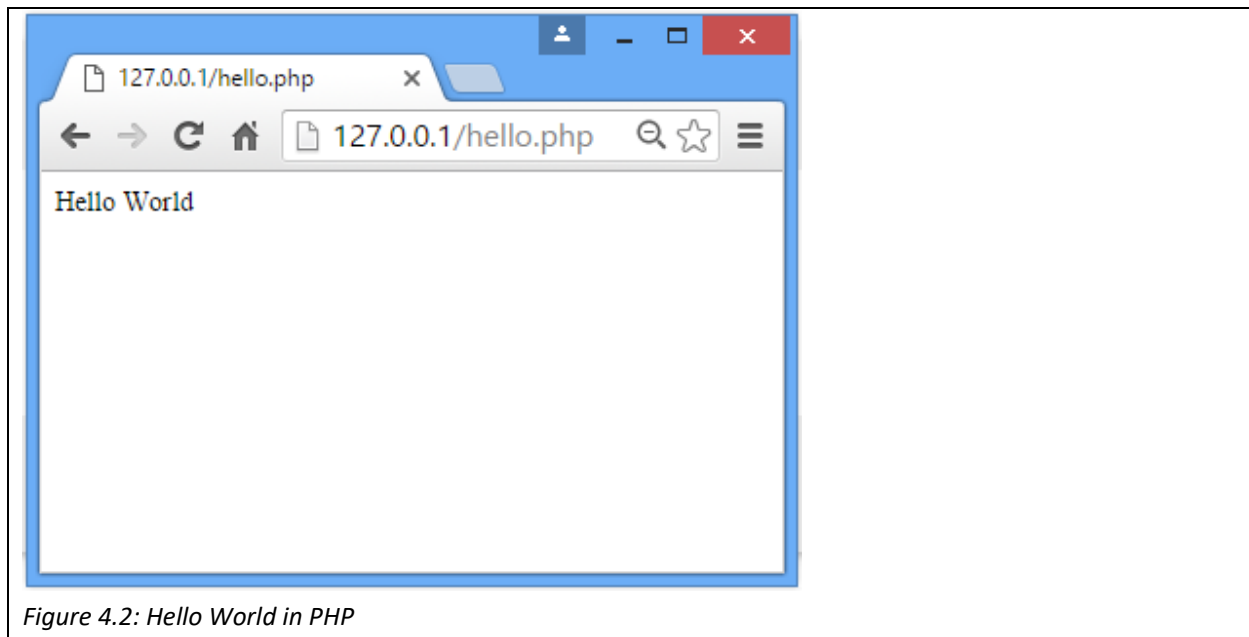


Figure 4.2: Hello World in PHP

4.2 PHP Basic Syntax

As shown in the opening “Hello World” example, PHP code is written inside the opening and closing PHP tags;

```
<?php           // Code Here           ?>
```


This could be substituted for simply an opening question mark tag, as follows, but it is suggested to keep using opening php tags;

```
<? // Code Here ?>
```

A block of PHP code can appear anywhere within the document, and it is interpreted by the server before the final HTML is returned to the browser. This can be seen if you inspect the source of the “Hello World” example. This also means that unlike HTML & JavaScript, the user can’t see what PHP code may have been executed to create the page. A PHP code block can contain HTML code, PHP code, or text. The “Hello World” example uses the PHP language construct ‘echo’ to display the words onto the document.

As with any programming language, comments are very useful to help programmers remember their reasoning, and also for any other developers who may need to maintain code in the future. Comments in PHP are left in the same style as in JavaScript or C++, using either ‘//’ or ‘/* ... */’, depending on whether the comment fits on a single line or spans multiple lines.

4.3 PHP Variables

Variables are containers that can be used in code to store some value. In PHP a variable name is preceded by the “\$” symbol, and the name, as with other programming languages, can be any combination of letters, numbers and the underscore character, so long as it doesn’t begin with a number. Variable names are case sensitive, so \$Name, \$name and \$NAME would all refer to different variables. PHP variables can contain a variety of different data types including; Strings, Integers, Float, Boolean, Array and Objects. In PHP variables are weakly typed, which means the type of the variable doesn’t need to be specified, PHP will automatically convert a variable depending on the value being stored there. In addition, the variable doesn’t need to be declared in advance, in PHP you can just start using variables.

```
<?php
  $x = 5;
  $y = "3.5";
  $z = $x + $y;
  echo "The number is $z<br>";
  echo "The number is " . $z . "<br>";
?>
```

This example demonstrates 3 variables, the first (\$x) stores an integer, while the second (\$y) stores a string. When the two are added together and stored in \$z, PHP has no problem with converting the data types, firstly from a string to a floating point number, and then adding a float to the integer. The resulting number is then displayed on the screen twice. The two echo statements produce the same result, as a variable can be included within an output string, or concatenated together using “.”.

As with other programming languages, a string is a sequence of characters, alphanumeric or symbols, which is contained inside quotes, either single quotes, or double quotes. Because strings are so important in web programming, PHP has plenty of string related functions, which will be discussed later. An integer is a whole number between -2,147,483,648 and 2,147,483,647; it can be positive or negative so long as it has no

decimal point. A variable containing a number with a decimal point is a float, otherwise known as double. A Boolean variable can take two values, either true or false.

Arrays are a special type of variable which can store multiple values. When data values are related it makes sense to store it in an array, rather than as separate variables. In PHP an array is created using the array() function. In this example an array of 3 student names is created and then the first student's name is output. By default PHP arrays are indexed beginning at 0. A further student name is then added to the array, which will have the index 3.

```
<?php
    $students = array("John", "Bob", "Steve");
    $students[] = "Fred";
    echo "The first student is " . $students[0];
?>
```

As well as indexed arrays, arrays can be associative, where the index isn't an integer. As with associative arrays in JavaScript, these arrays can have a string as the index.

```
<?php
    $ages = array("John"=>"18", "Bob"=>"19", "Steve"=>"22");
    echo "John is " . $ages["John"];
?>
```

Whether the array is indexed or associative, each element in the array is a key=>value pair. With an indexed array the key is an integer, while with an associative array the key is a string. The value can be of any type, including an array, which allows the creation of multi-dimensional arrays, or arrays of arrays. This example creates a 2 dimensional array to store students names, scores and grades.

```
<?php
    $students = array();
    $students[] = array("John", "84", "A");
    $students[] = array("Bob", "65", "C+");
    $students[] = array("Steve", "72", "B");
    echo "The first student's grade is " . $students[0][2];
?>
```

The string type of variable is particularly useful in web development, as many pages consist of strings. It is unsurprising that PHP includes a wide variety of functions to manipulate strings in different ways. The following table introduces just some of the string related functions included in PHP.

Function Name	Purpose
explode()	Splits a string into an array
implode()	Converts an array into a string
lcfirst()	Makes the first character of a string lowercase
str_replace()	Replaces specified characters in a string

str_word_count()	Returns the number of words in the string
strcmp()	Compares two strings
strlen()	Returns the length of a string
strpos()	Returns the position of the first occurrence of a string inside another string
strtoupper()	Converts a string to upper case
strtolower()	Converts a string to lower case
substr()	Returns a specified part of a string
ucfirst()	Makes the first character of a string uppercase

Table 4.1 String Related Functions in PHP

As mentioned this list is far from exhaustive, it just highlights some of the functions available in PHP.

4.4 PHP Operators

There are a variety of operators available in PHP.

Arithmetic Operators	Purpose	Example
+	Addition	\$x=1+2;
-	Subtraction	\$x=2-1;
*	Multiplication	\$x=1*2;
/	Division	\$x=2/1;
%	Modulus Returns the remainder from a division	\$x=2%1;
++	Increment (add one to the value)	\$x++;
--	Decrement (minus one from the value)	\$x--;

Table 4.2: Arithmetic Operators in PHP

The increment and decrement operators are again unary operators, which means they operate on one variable, whereas the other arithmetic operators operate on two values. The assignment operators are also similar assignment operators already encountered; the value from the expression to the right is assigned to the variable on the left. The notable addition is the “.” operator, which is used to concatenate strings.

Assignment Operators	Example	Equivalent
=	\$a = \$b;	
+=	\$a += \$b;	\$a = \$a + \$b;
-=	\$a -= \$b;	\$a = \$a - \$b;
*=	\$a *= \$b;	\$a = \$a * \$b;
/=	\$a /= \$b;	\$a = \$a / \$b;
%=	\$a %= \$b;	\$a = \$a % \$b;
.=	\$a .= \$b	\$a = \$a . \$b;

Table 4.3: Assignment Operators in PHP

PHP also has a selection of logical, or Boolean operators.

Logical Operators	Purpose
==	Equal to
!=	Not equal to
<>	Not equal to
<	Less than
>	Greater than

<=	Less than or equal to
>=	Greater than or equal to
&&	AND
	OR
!	Not

Table 4.4: Logical Operators in PHP

4.5 PHP Flow Control

Conditional statements are used in PHP to allow different actions to be performed in different circumstances. The first option is the “if-statement”, where subsequent actions are dependent on a Boolean expression. The “if statement” combines with the “elseif” and “else” statements to allow multiple different outcomes. These statements can be used to nest multiple conditions inside other conditions. Each case is tested by a Boolean expression, and if the condition is satisfied the code in the trailing braces is executed. In this example different messages are displayed on the screen depending on the value of the variable \$x.

```
<?php
if($x>20)
{
    echo "x is greater than 20";
}
elseif($x<10)
{
    echo "x is smaller than 20";
}
else
{
    echo "x is between 10 and 20";
}
?>
```

The alternative to using an “If” statement is to use a “switch” statement. The “switch” statement decides which action to perform based on the value of a variable. In the following example, the message depends on the value of the variable \$x. The “switch” statement is particularly useful for managing menus, and from this example a different page can be displayed in different conditions.

```
<?php
switch ($x) {
    case "login":
        echo "Display the Login Page";
        break;
    case "register":
        echo "Display the Register Page";
        break;
```

```
default:
    echo "Display the Guest Page";
    break;
}
?>
```

Looping, or iteration, is used to repeat the same block of code multiple times. As with other programming languages, PHP offers “while” loops to repeat a block of code based on a Boolean expression. The loop will repeatedly execute while the condition returns true. Care should be taken to avoid infinite loops and ensure the loop will eventually exit.

```
<?php
    $x = 10;
    while($x>0)
    {
        echo "There were " . $x . " in the bed, till someone said, roll over...<br>";
        $x--;
    }
?>
```

A “while” loop may never execute, if the initial Boolean expression returns false. A related alternative is the “do-while” loop, which will execute at least once as the Boolean test occurs after the code block.

```
<?php
    $x = 10;
    do {
        echo "There were " . $x . " green apples, ready to be eaten...<br>";
        $x--;
    } while($x>0);
?>
```

Both the “while” and “do-while” loop are particularly useful when the programmer doesn’t know how many times the loop will iterate, perhaps when waiting for some input from the user. An alternative form of iteration is the “for” loop, which is generally used when the programmer knows how many iterations should occur.

```
<?php
    for($i = 0; $i<10; $i++)
    {
        echo "I have " . $i . " doors, but need 1 more...<br>";
    }
?>
```

In a “for” loop the first line sets up the loop, in three parts. First the counter is initialized, followed by the Boolean test expression, and finally an update action, normally used to increment the counter. A further looping statement can be used with arrays in PHP. For various reasons we may not know the size of the array, and although PHP has a count() function which returns the size of an array, there are a variety of reasons for iterating through an array. The “foreach” loop provides an easy way of performing some action for every member of an array. The basic syntax for this loop is;

```
foreach($array as $value) { // code }
```

```
<?php
$count = 1;
foreach($students as $name)
{
    echo $count . ") Student name:- " . $name . "<br>";
    $count++;
}
?>
```

This example loops through an indexed array, outputting the name of each student. In each iteration the variable \$name takes on the value of each element in the array. The “foreach” loop can also work usefully with an associative array.

```
<?php
$count = 1;
foreach($students as $name => $grade)
{
    echo $count . ") Student " . $name . " got grade " . $grade . "<br>";
    $count++;
}
?>
```

With an associative array, both the key and the value contain useful information, so in this example, in each iteration the key is represented by the variable \$name, and the value is represented by the variable \$grade. Obviously the array would need to be set up correctly before the loop executes.

4.6 PHP Form Validation Example

Forms are often used to collect information from a user and send it to the server. As the user could input anything into the form, it is important to validate the inputs before processing it. Later we will examine using JavaScript for form validation in the browser before data is sent to the server, however it is still important to validate data in PHP, perhaps before inserting it into a database. In this example we will consider a simple form and check if the data entered satisfies a few conditions.

```
<form action="welcome.php" method="post">
  Name: <input type="text" name="name"><br />
  <input type="submit" value="Go!">
</form>
```

This form consists of just a text field for the user to input their name, and a submit button.

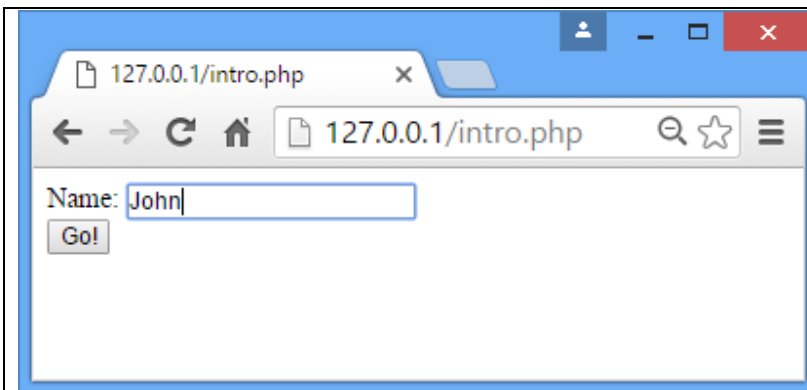


Figure 4.3: A Simple Form

The form tag contains 2 parameters, an action and a method. The action parameter states which page should be opened after the form is submitted – in this case a page called ‘welcome.php’, and the method parameter states the way data should be sent to the new page – in this case by POST. The next step is to create the “welcome.php” page.

```
<body>
  Welcome <?php echo $_POST["name"]; ?>
</body>
```

When the data is sent using the post method, the PHP superglobal array `$_POST` can be used to collect the data. In this case the `$_POST` array contains a member with key “name”, and the value the user filled in the form. Any values sent from the form will be stored in the `$_POST` superglobal array, so that subsequent PHP scripts can use the data. For example, at this point the PHP code could validate the form data, to make sure the user has submitted appropriate answers. The “welcome.php” page should look, as follows.

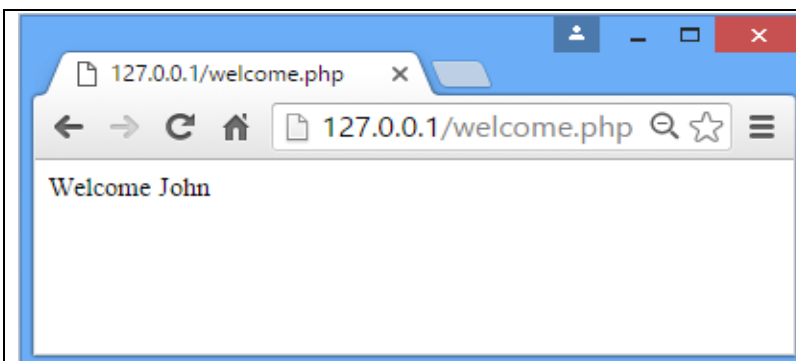


Figure 4.4: A Welcome Page using `$_POST`

The key alternative to posting data is to use the “GET” method. For the form, the only change needed is changing the method.

```
<form action="welcome.php" method="get">
  Name: <input type="text" name="name"><br />
  <input type="submit" value="Go!">
</form>
```

Similarly, welcome.php needs to be updated to collect it's data from the `$_GET` superglobal array.

```
<body>
  Welcome <?php echo $_GET["name"]; ?>
</body>
```

The following screenshot demonstrates a key difference between sending form parameters using the POST method, versus using the GET method. Notice the URL of the page includes the parameters and their values.

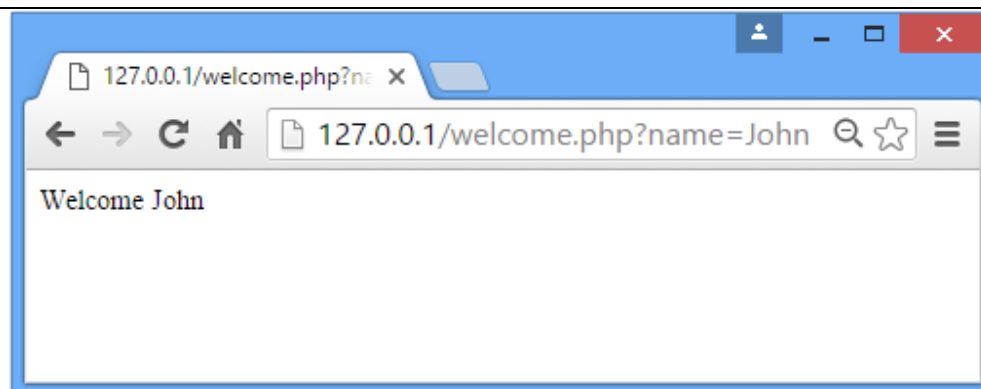


Figure 4.5: A Welcome Page using `$_GET`

There are several similarities, and several differences between the POST and GET methods. Both methods create an array of key=>value pairs containing the names of the form inputs and the values submitted. Both of these arrays are superglobals, accessible from any function, class or file on the page. The key difference demonstrated above is that the GET method passes the variables via URL parameters, and the POST method passes variables via the HTTP POST method. These differences are important when deciding whether to use the POST or GET method. The GET method is visible to everyone, which clearly has security issues, and should not be used for sensitive data, in this case the POST method is useful as the data is transferred invisibly. The URL is also limited to around 2000 characters, while there are no limits to how much data is sent via the POST method. One advantage to sending the variables in the URL is that the new page can be bookmarked, which isn't possible when data is sent using POST. Nonetheless generally using the POST method is the preferred method.

For testing purposes, PHP comes with a useful function that prints a human readable version of the array, so the developer can see what values have been successfully passed from the form. The function `print_r()` takes an array or object as it's parameter and displays the contents.

```
<body>
  <?php print_r($_POST); ?>
</body>
```

As an example for the POST array, the contents are as follows;

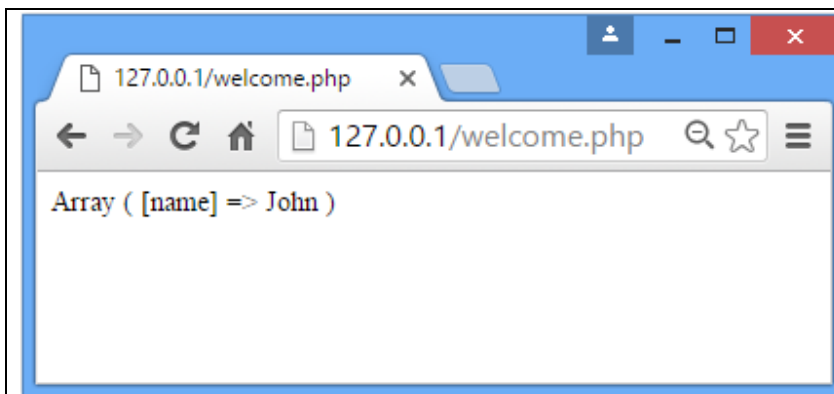


Figure 4.6: The POST Array

Once the form data has been posted to “welcome.php”, it could be validated. For example the name could be tested to ensure that it has at least 3 characters, and will output an error message if it doesn't.

```
<body>
  <?php
    $name = $_POST["name"];
    if(strlen($name)<3)
    {
      echo "Name must be at least 3 characters.";
    }
  ?>
</body>
```

The form could be extended to add further fields, such as gender, and password, and these could also be validated using PHP scripts once the form has been submitted to the server. The following file “intro.php”, has 3 sections to the form.

```
<form action="welcome.php" method="post">
  Username: <input type="text" name="username"><br />
  Gender:<br />
  <input type="radio" name="gender" value="male" /> Male
  <input type="radio" name="gender" value="female" /> Female<br />
```

```
Password:<input type="password" name="pass1" /><br />
Re-enter Password:<input type="password" name="pass2" /><br />
<input type="submit" value="Go!">
</form>
```

When the form is submitted the data is sent to the file “welcome.php”, which can perform some basic validation.

```
<body>
<?php
$username = $_POST["username"];
if(strlen($username)<3)
{
    echo "Name must be at least 3 characters.<br>";
}
if(array_key_exists("gender", $_POST))
{
    $gender = $_POST["gender"];
}
else
{
    echo "Gender must be specified.<br>";
}
$pass1 = $_POST["pass1"];
$pass2 = $_POST["pass2"];
if($pass1 == "")
{
    echo "Password can not be blank.<br>";
}
if($pass1 != $pass2)
{
    echo "Passwords must match.<br>";
}
?>
</body>
```

Consider briefly the differences between validating form inputs using JavaScript in the browser, versus doing it using PHP. Performing form validation using JavaScript is useful as it traps any errors before the form is sent to the server, which means the user won't need to reload the page to fill the form in again. Therefore using JavaScript for form validation is likely to improve the experience of the user. Validating the form using JavaScript means that the data isn't sent to the server until it satisfies the requirements, putting more work on

the client machine, and less load on the network and server. However, JavaScript is not a substitute for server side validation – as the JavaScript is downloaded and run on the user’s machine and so visible to an experienced user. As we will see in later chapters, PHP validation is also important, certainly before allowing inputs to be inserted into a database.

Key Points

- PHP is an important Server Side programming language.
- Before running PHP code a web server needs to be set up to process PHP code.
- When a web server receives a request for PHP, first it processes the code before returning HTML.
- Variables in PHP are weakly or loosely typed and can also be used to store arrays.
- A form can be used to send data to the server using the post method.
- Data sent via the post method then exists within the `$_POST` superglobal array, or using the get method to the `$_GET` superglobal array.
- The main difference between post and get is that when using get, the parameters are attached to the url, so that it can be bookmarked.

Further Resources

- 1) The full documentation and complete function reference guide to PHP can be found at:-
<http://php.net/manual/en/>
- 2) The w3schools tutorial for PHP can be found here:-
<https://www.w3schools.com/php/>
- 3) A beginners guide to PHP development can be found here:-
<https://www.tutorialspoint.com/php/>

Assignment

Extend the basic form validation script! Create a form to register for a website. Your script should validate the following inputs;

- 1) Forename – Must not be blank, must not contain spaces, and must have at least 3 alphabet characters
- 2) Surname – Same rules as for Forename
- 3) Username – At least 5 characters and can include numbers, _ and –
- 4) Password – Must be at least 8 characters, containing both upper and lower case letters, numbers and symbols.
- 5) Age – The age must be between 18 and 110
- 6) Email – Must be of the form abc@def.ghi

Enhance your form to make sure the user re-inputs their password the same twice and hide the input from the view.

Chapter 5

PHP Functions and Objects

Objectives

This chapter extends the introduction to PHP by looking closer at functions and objects. Some of the key pre-defined functions in PHP are introduced, notably those related to arrays, mathematics and date / time functions. The chapter also examines how programmers can create their own functions and why they would want to. Objects are also briefly introduced, before a demonstration of how a simple calendar can be built. This calendar will be extended in later chapters. After reading this chapter you should:-

- Be comfortable with calling existing PHP functions.
- Be aware of some of the key existing functions (related to arrays, mathematics, and dates and time).
- Be able to define your own PHP functions.
- Be able to store functions in a separate file and include them in your project.
- Know how to use include and require statements.
- Know how objects can be used in PHP.

Contents

- 5.1 Array Functions
- 5.2 Mathematical Functions
- 5.3 Date and Time Functions
- 5.4 Defining Your Own Functions
- 5.5 include and require
- 5.6 Objects in PHP
- 5.7 Creating a Calendar

The last chapter introduced the basics of programming in PHP, in this chapter we look closer at functions and objects in PHP. Functions are a key component of coding, resulting in less typing, simpler syntax, fewer errors, quicker loading and execution time, and importantly simpler logic. If code isn't broken down sensibly into functions, it quickly grows to a state where it is difficult to maintain. PHP has hundreds of functions built in, and the previous chapter introduced some of the string related functions available to PHP developers. Existing PHP functions can be called from anywhere in the code, a function call just needs to reference the function name, and then contain any necessary parameters, or arguments, within brackets ().

```
<?php
echo strrev(".dlroW olleH");
str_repeat("Hip", 2);
$result = strtoupper("hooray!");
echo ucfirst(strtolower("jOhN"));
?>
```

These examples illustrate some of the string functions. The first function `strrev()` takes 1 parameter, a string, and it will reverse the string and output "Hello World.". The second function, `str_repeat()`, takes 2 parameters, and will repeat the string 2 times, it also demonstrates how a function can be called without the result being used. The third function, `strtoupper()`, again only takes 1 parameter, but will convert the string parameter to uppercase returning "HOORAY!". This time the result is stored in a variable called `$result`. Finally the last line makes 2 functions calls. Firstly the `strtolower()` function is called, converting the string parameter to lower case "john". The function then returns the new string as a parameter to the `ucfirst()` function, which makes the first character upper case, leaving the ultimate result as "John".

As developers gain experience they gain find uses for more existing functions, and with the wide variety of existing functions available, when faced with a new problem it is worth spending time to look for and use an existing solution before writing a new solution from scratch. There are books dedicated to listing PHP functions in detail, as well as the excellent resources at php.net. The first part of this chapter will introduce some of the more useful functions that PHP offers, before later discussing how to create your own functions.

5.1 Array Functions

As seen in the previous chapter, an array is a very useful type for storing a collection of data. PHP offers a variety of functions for manipulating an array. This section introduces some of the more useful array related functions.

5.1.1 `array()`

`array()` is used to create an array, and can be supplied with parameters to initialize the values in the array. In this example first an array of 4 elements is created, with indexes from 0-3. The second array creates an associative array listing 4 country codes and their respective countries.

```
<?php
$arr = array("ant", "bird", "cat", "dog");
$arr2 = array("uk"=>"United Kingdom", "th"=>"Thailand", "kh"=>"Cambodia", "za"=>"South Africa");
?>
```

5.1.2 `print_r()`

`print_r()` is a function that developers can use to output a human readable version of a variable, in this case it will display the contents of the array. It takes one parameter, the name of the array. This example displays the two arrays created above.

```
<?php
print_r($arr);
echo "<br>";
print_r($arr2);
?>
```

This would display the following.

```
Array ( [0] => ant [1] => bird [2] => cat [3] => dog )
Array ( [uk] => United Kingdom [th] => Thailand [kh] => Cambodia [za] => South Africa )
```

Figure 5.1: `print_r()` in PHP

5.1.3 `array_count_values()`

This function counts the number of times each value occurs in the array. The function returns a new array which matches the value as the key, and the frequency as its value.

```
<?php
$arr = array("ant", "bird", "ant", "dog");
print_r(array_count_values($arr));
?>
```

This example creates an array of 4 elements, however the word "ant" appears twice, so the result of calling `array_count_values()` is as follows;

```
Array ( [ant] => 2 [bird] => 1 [dog] => 1 )
```

Figure 5.2: `array_count_values()` in PHP

5.1.3 `array_diff()` & `array_intersect()`

This function computes the difference between two (or more) arrays. It takes at least 2 arrays as parameters, and returns an array containing any values that occur in the first array, but not in any other array.

```
<?php
$arr = array("ant", "bird", "cat", "dog");
$arr2 = array("ant", "bird", "ant", "dog");
print_r(array_diff($arr, $arr2));
?>
```

In this example, the string “cat” appears in the first array, but not the second array, so the result of calling `array_diff()` is a new array that only contains “cat”.

```
Array ( [2] => cat )
```

Figure 5.3: `array_diff()` in PHP

Similar to `array_diff()`, PHP also has a function called `array_diff_assoc()`, where not only the value is considered, but also returns array elements that don't have the same key. While `array_diff()` is used to identify the differences between 2 arrays, `array_intersect()` can be used to find the similarities between 2 arrays. This time the returned array contains all the elements which are found in the second array.

```
<?php
$arr = array("ant", "bird", "cat", "dog");
$arr2 = array("ant", "bird", "ant", "dog");
print_r(array_intersect($arr, $arr2));
?>
```

This would output.

```
Array ( [0] => ant [1] => bird [3] => dog )
```

Figure 5.4: `array_intersect()` in PHP

Note that there is also a function called `array_intersect_assoc()`, which computes the intersection of 2 arrays, but also considers the index as well as the value.

5.1.4 `array_flip()`

`array_flip()` will swap each key with its value, so that the keys become the values, and the values become the keys.

```
<?php
$arr = array("uk"=>"United Kingdom", "th"=>"Thailand", "kh"=>"Cambodia", "za"=>"South Africa");
print_r(array_flip($arr));
?>
```

The resulting output from this example would be.

```
Array ( [United Kingdom] => uk [Thailand] => th [Cambodia] => kh [South Africa] => za )
```

Figure 5.5: `array_flip()` in PHP

5.1.5 `in_array()` & `array_key_exists()`

It is often useful to check whether an element exists in the array before performing some task on it. `in_array()` tests if a given value occurs in the array and returns true if it is found.

```
<?php
$arr = array("uk"=>"United Kingdom", "th"=>"Thailand", "kh"=>"Cambodia", "za"=>"South Africa");
if(in_array("Thailand", $arr))
{
    echo "That exists!";
}
?>
```

Related to `in_array()`, a further function, `array_key_exists()`, returns a Boolean, true if the array contains the key, and false if it doesn't.

```
<?php
$arr = array("uk"=>"United Kingdom", "th"=>"Thailand", "kh"=>"Cambodia", "za"=>"South Africa");
if(array_key_exists("th", $arr))
{
    echo "That exists!";
}
?>
```

5.1.6 `array_reverse()`

This function, unsurprisingly, will reverse the elements in the array.

```
<?php
$arr = array("ant", "bird", "cat", "dog");
print_r(array_reverse($arr));
?>
```

The result of this code is.

```
Array ( [0] => dog [1] => cat [2] => bird [3] => ant )
```

Figure 5.6: `array_reverse()` in PHP

5.1.7 *count()*

This function simply returns the number of elements in the array. It has an alias, *size_of()*, which performs exactly the same thing. If the parameter isn't an array (or a countable object), then 1 will be returned. The *is_array()* function can be used to check if the parameter is an array.

```
<?php
  $arr = array("ant", "bird", "cat", "dog");
  if(is_array($arr))
  {
    $size = count($arr); // $size == 4
  }
?>
```

5.1.8 *shuffle()*

The shuffle function will randomize the order of the elements in the array. Note that the function will return a Boolean where true represents that the function has successfully completed.

```
<?php
  $arr = array("ant", "bird", "cat", "dog");
  shuffle($arr);
  print_r($arr);
?>
```

Note that this function will produce different results each time it is executed, but one possible result is as follows.

```
Array ( [0] => cat [1] => bird [2] => dog [3] => ant )
```

Figure 5.7: *shuffle()* in PHP

5.1.9 *Sorting Arrays*

PHP comes with several sorting functions, depending on exactly how the array should be sorted, i.e. whether it should be sorted based on the value, or the key, whether it should be sorted in ascending or descending order, and whether the relationship between the keys and their values should be maintained. Table 5.1 lists 6 functions, and details how they sort an array. The basic sort function is *sort()*, which can be done in reverse order using *rsort()*. Unlike *sort()*, *asort()* maintains the relationship between a key and its value, and can be done in reverse using *arsort()*. All four of those functions sort the array based on the value, while *ksort()* can be used to sort an array based on its key, with *krsort()* performing a reverse sort based on the key.

Name	Sort By	Order	Maintain Key=>Value Association
sort()	Value	Low to high	No
rsort()	Value	High to low	No
asort()	Value	Low to high	Yes
arsort()	Value	High to low	Yes
ksort()	Key	Low to high	Yes
krsort()	Key	High to low	Yes

Table 5.1: Sorting Arrays in PHP

Let's look at the results of sorting an array using the different approaches, beginning with sort().

```
<?php
$arr = array("uk"=>"United Kingdom", "th"=>"Thailand", "kh"=>"Cambodia", "za"=>"South Africa");
sort($arr);
print_r($arr);
?>
```

This would result in the following order, notice how the association between the keys and values has been lost.

```
Array ( [0] => Cambodia [1] => South Africa [2] => Thailand [3] => United Kingdom )
```

Figure 5.8: Sorted Array in PHP

The following shows all 6 different types of sort.

```
sort() - Array ( [0] => Cambodia [1] => South Africa [2] => Thailand [3] => United Kingdom )
rsort() - Array ( [0] => United Kingdom [1] => Thailand [2] => South Africa [3] => Cambodia )
asort() - Array ( [kh] => Cambodia [za] => South Africa [th] => Thailand [uk] => United Kingdom )
arsort() - Array ( [uk] => United Kingdom [th] => Thailand [za] => South Africa [kh] => Cambodia )
ksort() - Array ( [kh] => Cambodia [th] => Thailand [uk] => United Kingdom [za] => South Africa )
krsort() - Array ( [za] => South Africa [uk] => United Kingdom [th] => Thailand [kh] => Cambodia )
```

Figure 5.9: All 6 Different Types of Sort in PHP

5.2 Mathematical Functions

Much of coding involves performing calculations, and fortunately PHP has a wide selection of math-related functions. This section introduces some of those functions.

5.2.1 abs()

Finding the absolute value, or modulus of a number can be done simply using the abs() function. It will take a number, either integer or floating point, and return the non-negative version.

```
<?php
echo abs(-3.4); //outputs 3.4
echo abs(3.4); //also outputs 3.4
?>
```

5.2.2 *ceil()*, *floor()* and *round()*

Rounding a floating point number to an integer can be done in a couple of ways. Using `ceil()`, the number will be rounded up, while using `floor()`, the number will be rounded down. Alternatively `round()` can be used to round a number to the nearest integer, depending on whether the floating point is less than 0.5, or not. `round()` can also take a second parameter which specifies the number of decimal places to round a number to.

```
<?php
echo ceil(-3.4); //outputs 4
echo floor(3.4); //outputs 3
echo round(3.5); //outputs 4
echo round(3.5446, 2); //outputs 3.54
?>
```

5.2.3 *log()* and *exp()*

The natural logarithm for a number can be found using the `log()` function, and conversely the exponent can be calculated using `exp()`.

```
<?php
echo log(5); //outputs 1.6094379124341
echo exp(5); //outputs 148.41315910258
?>
```

5.2.4 *max()* and *min()*

These two functions can accept an array as a parameter, and `max()` will return the highest value in the array, while `min()` will return the lowest.

```
<?php
$arr = array(4, 5, 3, 7, 2);
echo max($arr); //outputs 7
echo min($arr); //outputs 2
?>
```

5.2.5 *pi()*

The `pi()` function will return an approximation for pi, generally set to 14 decimal places, although the precision can be adjusted in the PHP initialization file, 'php.ini'.

```
<?php
echo pi(); //outputs 3.1415926535898
?>
```

5.2.6 pow() and sqrt()

PHP offers functions for managing powers. The pow() function takes two parameters, with the base being the first parameter and the power being the second parameter. Square roots can also be calculated using the sqrt() function.

```
<?php
echo pow(2,3); //outputs 8 (2^3)
echo sqrt(25); //outputs 5
?>
```

5.2.7 Random Numbers

PHP offers way to generate random numbers. Earlier versions of PHP required the use of srand() to seed a random number sequence, but since PHP version 4.2.0 this is done automatically. Random numbers can therefore be generated using rand(). The rand() function can take 2 parameters, one for the lowest number in the range and the second for the highest number in the range. Whilst the rand() function has been generating random numbers for a long time, the Mersenne Twister algorithm has improved random number generation, both in terms of speed and entropy. So, an alternative is the mt_rand() function.

```
<?php
echo rand(10,50); //outputs a number between 10 and 50.
echo mt_rand(10,50); //outputs a number between 10 and 50, generated using Mersenne Twister
?>
```

5.2.8 Trigonometry

PHP has a variety of function to support trigonometry, as listed here.

Name	Performs
cos()	Cosine
cosh()	Hyperbolic Cosine
acos()	Arc Cosine
acosh()	Inverse Hyperbolic Cosine
sin()	Sine
sinh()	Hyperbolic Sine
asin()	Arc Sine
asinh()	Inverse Hyperbolic Sine
tan()	Tangent
tanh()	Hyperbolic Tangent
atan()	Arc Tangent
atanh()	Inverse Hyperbolic Tangent

Table 5.2: Trigonometry Functions in PHP

5.3 Date and Time Functions

PHP also offers a variety of functions that help manipulate times and dates. The first thing to note is the difference between Date() in PHP and Date() in JavaScript, with the main difference being where the function is run – while the client side JavaScript is run on the client machine, the PHP version runs on the server, so the results may be different depending where in the world the machine is located. The first concept to introduce is the PHP time() function. This function will return an integer, which counts the number of seconds since the Unix Epoch. The Unix Epoch was midnight at the start of January 1, 1970 in the UTC timezone, so time() returns a timestamp, or in other words the number of seconds since then.

In 2016, there had been around 1.4 billion seconds since the Unix Epoch, so clearly working with a large integer like this isn't convenient. Fortunately PHP offers a variety of functions that can make time manipulation simpler. The date() function can be used to reformat the timestamp, to produce a string that can display dates in different ways, depending on the parameters sent to it. There is a lot of flexibility for formatting dates, but here are some examples. These three examples will output the same timestamp (now), in different formats.

```
<?php
echo date("d/m/Y") . "<br>"; //outputs the date for example:- 24/07/2016
echo date("h:i:sA") . "<br>"; //outputs the time for example:- 01:09:58PM
echo date("l jS M, Y") . "<br>"; //outputs a date for example:- Sunday 24th July, 2016
?>
```

There are a wide variety of parameters that can be used to change the format of the date string, depending on the characters used. Note that the character is case sensitive, and lower case characters may return a different result to upper case characters.

Character	Description	Example
d	2 digit day of the month	01 to 31
D	3 letter text representation of day	Sun to Sat
j	Day of month without leading zero	1 to 31
l	Full day name	Sunday to Saturday
N	Number of day of the week	1 (Monday), to 7 (Sunday)
S	2 character suffix for day of the month	st, th, nd, or rd
z	Day of the year	0-365
W	Week number in the year	0-52
F	Name of the month	January – December
m	Number of the month	01-12
M	3 letter representation of the month	Jan – Dec
n	Number of month, without leading zero	1-12
t	Number of days in the month	28,29,30,31
L	Boolean 1 if a leap year, 0 if not.	1 or 0
Y	4 digit representation of year	2016 or 2001
y	2 digit representation of year	16 or 01
a	Lower case am or pm	am or pm
A	Upper case AM or PM	AM or PM

g	Hour, in 12 hour format	1-12
G	Hour, in 24 hour format	0-23
h	Hour, in 12 hour format, with leading zero	01-12
H	Hour, in 24 hour format, with leading zero	00-23
i	Minutes	00-59
s	Seconds	00-59
T	Timezone abbreviation	EST, GMT

Table 5.3: Parameters for the PHP date() Function

As can be seen from the examples above there is great flexibility in how to display the date, and realizing that dates are stored as an integer, representing the number of seconds means that calculations can be made when dealing with different dates.

5.4 Defining Your Own Functions

While PHP has many useful functions available for developers to use, it is inevitable that more functions will be needed to satisfy specific problems, and to sensibly break down larger problems into manageable ones. Defining functions in PHP is also relatively straight forward. The general syntax for defining a new function is as follows.

```
function functionname(parameter1, parameter2)
{
    //Statements
}
```

A value could be returned from the function as shown in the following example.

```
<?php
function average($v1, $v2, $v3)
{
    return ($v1 + $v2 + $v3)/3;
}
?>
```

Given that the value returned from a function could be of any type, that type could be an array (or an object), which means that multiple values could be returned from one function.

```
<?php
function average()
{
    $v1 = 4;
    $v2 = 5;
    $v3 = 6;
    return array($v1, $v2, $v3)
}
?>
```

Reference parameters are also possible in PHP, by simply attaching an ampersand “&” to the parameter during the function definition. Regular, value, parameters will make a copy of any data that is sent to the function, while with reference parameters a reference to the memory location of the data is used, which means the same data is used in the function as from where it is called. The following example demonstrates the concept of reference parameters. The function called increment is defined with 2 parameters, the first is a regular call by value parameter, while the second is a call by reference parameter. In the function, both parameters are incremented. The demonstration shows how the first parameter remains the same before and after the function is called, while the second parameter increases.

```
<?php
  $x = 3;
  $y = 4;
  echo "x = " . $x . " & y = " . $y . "<br>"; //outputs "x = 3 & y = 4"
  increment($x, $y);
  echo "x = " . $x . " & y = " . $y . "<br>"; //outputs "x = 3 & y = 5"
  function increment($v1, &$v2)
  {
    $v1++;
    $v2++;
  }
?>
```

As can be seen in this example, functions can be written anywhere within the PHP files, and it may make sense to define the functions at the bottom of the page so they are moved away from the rest of the code. Alternatively, it makes more sense to move functions to a separate file. Overtime more useful functions are created, so they could be moved to a separate file, perhaps called “functions.php”. The functions file can then be included on any page that needs it with the following “include” statement, using the appropriate relative url to identify the file.

```
<?php
  include "functions.php";
?>
```

5.5 Include and Require

As a project grows, it may well make sense to break parts of each page into multiple files, each of which can be included using the include statement. Consider a website where every page has the same header, the same navigation panel, and the same sidebar, and only the content of a main panel is different. In this case each page could simply include the header, and include the navigation section. In the next chapter we will explore how functions required to connect to a database can be separated.

```
<?php
include "functions.php";
include "header.php";
include "navigation.php";
include "side-bar.php";
// Main Content Here
include "footer.php";
?>
```

Suppose a change is needed to the navigation pane, or to the sidebar, and that change needs to be made throughout the whole website. Using the techniques we have discussed so far, all pages would need to be edited, whereas if the navigation is separated into a single file, "navigation.php", any changes would only need to be made in a single place to affect the whole website. The "include" statement is a very useful part of PHP, and is closely related to another statement "require". The difference between 'include' and 'require' appears when the file can't be found. If the file doesn't exist, attempting to include it using the include statement will produce a warning, while with require it will produce a fatal error. Besides from that, the require statement is very similar to the include statement.

```
<?php
require "functions.php";
?>
```

As a project grows, and potentially includes multiple developers, another scenario could potentially arise. Multiple files could end up including multiple other files, where file A could include file B and file C, while file B also includes file C. This could lead to naming problems if the same function names are used in multiple locations. Fortunately this could be simply resolved using "include_once", or "require_once", which is essentially the same, except it ensures any file will only be included once.

```
<?php
include_once "functions.php";
require_once "database.php";
?>
```

5.6 Objects in PHP

PHP also supports the creation of objects, where the developer can create new types of data, and define how that data can be used. Objects are defined in a class, which defines both the data parts of the object and any member functions, or methods, that can be called for a function. The following creates a simple fraction class that consists of a numerator and a denominator, and has a function to display the fraction. This book won't dive deeply into the concepts of Object Oriented Programming (OOP), but this example demonstrates a simple class, as more objects will be encountered in coming chapters. In this example, both the method and the variables are public, which means they can be accessed from anywhere in the code.


```

<?php
class fraction
{
    public $num, $denom;
    function display()
    {
        echo $this->num . "/" . $this->denom;
    }
}
?>

```

This means an object can be initiated, which in PHP uses the new keyword. The members can then be accessed using the arrow operator “->”.

```

<?php
$f1 = new fraction;
$f1->num = 1;
$f1->denom = 2;
$f1->display();
?>

```

Further OO concepts can be implemented in PHP. Constructor functions are automatically called when an object is initiated, using the new keyword. A constructor function can be created using the __construct function.

```

<?php
function __construct()
{
    $this->num = $this->denom = 1;
}
?>

```

The example given above makes the members public, but protected and private are also valid, where private members can only be accessed through an interface created by the class, and protected members can be accessed from within the class or subclasses. This shows that PHP also supports inheritance where one class can extend another class.

5.7 Creating a Calendar

In this next section we put together several of the topics covered so far, to create a very simple calendar. This version of a calendar provides a form for the user to create an appointment, and once the appointment has been created a calendar is displayed with the appointment on display. There are many ways that the calendar can be developed further, using techniques that will be discussed later in the book. The

calendar begins with a simple form with just 2 input fields, one for the date, and one for the title of the appointment. Note that css can be used to make the form look more attractive.

```
<form action="calendar.php" method="post">
  Date: <input type="date" name="date"><br />
  Title: <input type="text" name="title" /><br />
  <input type="submit" value="Go!">
</form>
```

Note that input for the date uses the input type 'date', which provides a drop down box to easily choose a date. The form action is to send the data to 'calendar.php', using the post method. The main calendar file will then display a calendar that looks something like the image below.

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
					1	2
3	4	5	6	7	8	9
10	11	12	13	14 Meeting	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Figure 5.10 A Calendar in PHP

The month is broken down into days, with each day presented on a grid. This particular month (July 2016) began on a Friday, so there were 5 blank days before the 1st appears on Friday. Similarly at the end of the month there are 6 blank days as the month ended on a Sunday. The appointment, “Meeting”, was made for Thursday 14th. Also today (26th) is highlighted.

The first part of the calendar.php file is used to create variables storing the data needed to draw a calendar, this data includes which month of which year is being displayed, and how many days there are in that month. The first day of the month is also calculated, as well as today’s date, (day, month and year). To do this some of the date related functions discussed in this chapter are used as illustrated in the following code sample.

```
<?php
    $day = date('d', strtotime($_POST['date']));           //Gets day of appointment (1-31)
    $month = date('m', strtotime($_POST['date']));       //Gets month of appointment (1-12)
    $year = date('Y', strtotime($_POST['date']));        //Gets year of appointment (e.g. 2016)
    $firstday = date('w', strtotime('01-' . $month . '-' . $year)); //Gets the day of the week for the 1st of
                                                         //the month. (e.g. 0 for Sun, 1 for Mon)

    $days = date('t', strtotime($_POST['date']));       //Gets number of days in month
    $title = $_POST['title'];                           //Gets appointment title
    $today = date('d');                                 //Gets today’s date
    $todaymonth = date('m');                            //Gets today’s month
    $todayyear = date('Y');                             //Gets today’s year
?>
```

Looking closer at the calendar, each day consists of a square, surrounded by a black border. The column headers are a different height, but the rest of the days are nearly the same. The main difference (apart from the number) is that the blank days are shaded, today is highlighted. These all concern the style of each box, so this of course can be handled by css. In the following code block a number of selectors are created, beginning with a wrapper for the whole calendar, and then selectors for different kinds of boxes.

```
.calendar{
    position:relative;
    width: 960px;
}
div.date, div.days{
    width: 120px;
    border: 1px solid black;
    float: left;
    margin: 1px;
}
.blankday{
    background:#ccc;
}
```

```
div.date{
  height: 120px;
}
.today{
  background:#cfc;
}
```

The top row of the calendar shows the names of each day, which can use the days selector to create a box with width 120px for each day.

```
<div class="calendar">
  <div class="days">Sunday</div>
  <div class="days">Monday</div>
  <div class="days">Tuesday</div>
  <div class="days">Wednesday</div>
  <div class="days">Thursday</div>
  <div class="days">Friday</div>
  <div class="days">Saturday</div>
```

PHP code can then be used to display the rest of the calendar. This is done in 3 parts – first a number of blank days are created, depending on which day of the week the month begins. Each box is given the selectors date and blankday.

```
<?php
for($i=1; $i<=$firstday; $i++)
{
  echo '<div class="date blankday"></div>';
}
?>
```

Secondly the actual days of the month can be displayed, using a for loop to loop through each day in the month. Each box is given the 'date' selector, but if the day matches today the 'today' selector is added. As the variable \$i is incremented it can be displayed in the box. Finally if the day matches the appointment, the title of the appointment is also added.

```
<?php
for($i=1; $i<=$days; $i++)
{
  echo '<div class="date';
  if ($today == $i && $todaymonth==$month && $todayyear == $year)
  {
    echo ' today';
  }
}
```

```

echo "' . $i . '<br>';
if($day==$i)
{
    echo $title;
}
echo '</div>';
}
?>

```

Finally the remaining blank days can be displayed, after a quick calculation to find how many extra days are needed at the end of the month.

```

<?php
$daysleft = 7-((($days + $firstday)%7);
if($daysleft<7)
{
    for($i=1; $i<=$daysleft; $i++)
    {
        echo '<div class="date blankday"></div>';
    }
}
?>

```

Key Points

- PHP has many existing functions that can be called to make coding simpler.
- Arrays are a useful way of managing collections of related data.
- PHP has several existing functions to assist with managing arrays, including several sorting functions.
- All common mathematical functions exist within PHP.
- Time is counted as an integer for the number of seconds since the Unix epoch, but to make life easier PHP has functions for formatting the time and date in different ways.
- Although PHP has plenty of existing functions, good programmers break their code down into more functions.
- The include and require statements are another great way of dividing up a larger project, by allowing the programmer to include any files that are needed.
- Objects allow developers to create new types of data by combining data and behavior.

Further Resources

- 1) php.net provides a complete reference guide to the existing PHP functions, which can be found here:- <http://php.net/manual/en/funcref.php>

- 2) php.net also provides a guide to creating your own functions, found here:-
<http://php.net/manual/en/language.functions.php>
- 3) w3schools offers both a guide to existing PHP functions, and a guide to creating more functions, here:-
https://www.w3schools.com/php/php_functions.asp
- 4) A further tutorial to PHP functions can be found at tutorialspoint.com, here:-
https://www.tutorialspoint.com/php/php_functions.htm
- 5) KillerPHP has a beginners guide to creating and using objects in PHP, here:-
<https://www.killerphp.com/tutorials/php-objects-page-1/>

Assignment

This chapter includes a tutorial for creating a simple, basic calendar. Once you've built the basic calendar, use CSS to make it look more attractive. Add links to allow the user to move from month to month. This calendar offers a month view, extend it to add a week view and a day view that users can use to get more details on their screen – make sure your calendar is user friendly, so users can understand how to use it without being taught!

Chapter 6

PHP Files & MySQL Databases

Objectives

This chapter continues to expand writing code in PHP to run on the server, this time focusing on how to store data on the server. First files are introduced, explaining how PHP can manipulate files on the server, including managing security issues. An important part of the WAMP stack is MySQL, which refers to the database on the server where data is stored, so the second half of this chapter focuses on MySQL and how PHP is used to manage this data. After reading this chapter you should:-

- Be able to read, write and append files stored on your server by using PHP code.
- Understand `chmod()` and how to manage access permissions for files.
- Be able to use `phpmyadmin` to create and manage databases on your server.
- Understand the basics of Structured Query Language (SQL) and how it can be used to retrieve, insert, update and delete data from the database.
- Know how the `MySQLi` extension is used to allow PHP to run SQL queries.
- Understand both the procedural and object-oriented approaches to `MySQLi`.

Contents

- 6.1 PHP and Files
- 6.2 Introducing MySQL and `phpmyadmin`
- 6.3 Structured Query Language
- 6.4 `MySQLi`

At the end of the previous chapter, a simple calendar was displayed with an appointment. This demonstrated again how forms can be used to pass data from one page to another, but unfortunately the data was only stored temporarily in variables, which means if the browser is closed and reopened, all the data will be lost. Fortunately PHP offers easy ways of storing data permanently and in this chapter we will investigate storing data in files, or in a database. PHP offers convenient ways of storing data in files on the server, but often a better way is to use the MySQL database, which is part of the WAMP stack, and has a good interface with PHP.

6.1 PHP and Files

PHP offers a selection of functions useful for reading and writing from files stored on the server. We begin with a simple example that writes “Hello World” to a file.

```
<?php
$file = fopen("test.txt", "w");
fwrite($file, "Hello World");
fclose($file);
?>
```

This example calls 3 file related functions, ‘fopen()’, ‘fwrite()’ and ‘fclose()’. If you open this example in your browser, nothing will appear, but checking in the same directory a new file will have been created, called ‘test.txt’, and this file will contain the text “Hello World”. The first function is fopen(), to create a handle that is connected to a file. The function takes 2 parameters, the first being the name of the file, and the second being the mode that the file is opened, in this case the mode is “w”, for ‘write’. There are several different modes by which a file can be opened.

Mode	Description
r	Read only. Starting at the beginning of the file
r+	Read / Write. Starting at the beginning of the file
w	Write only. Clears the file if it isn't empty, or creates a new file if it doesn't exist
w+	Read / Write. Clears the file if it isn't empty, or creates a new file if it doesn't exist
a	Append. Writes to the end of a file, or creates a new file if it doesn't exist
a+	Read / Append. Keeps file content and writes to the end of the file
x	Write only. Creates a new file, and returns an error if the file exists
x+	Read / Write. Creates a new file, and returns an error if the file exists

Table 6.1: PHP File Open Modes

Once the file has been opened in one of the modes listed above it can be written to, or read from, obviously dependent on the mode. Matching the fopen() function, is the fclose() function. While fopen() creates a handle which acts as a connection to an open file, fclose() closes that handle when the file is no longer needed. It might be tempting to forget about fclose(), as when the script ends, PHP will automatically close the connection for you. However, on top of it being good programming practice to always close file connections, there are several reasons to. The file may be locked by the script until the end, which may prevent other scripts from accessing it, or PHP may crash leaving an open connection, or as code evolves in the future what was

once a safe exit, may become a problem. Simply put, match the `fopen()` function with a `fclose()` function when the handle is no longer needed.

As demonstrated in the previous example, writing to a file is done using `fwrite()`, which takes two parameters, firstly a file handle (as created with `fopen()`) and secondly the string that should be written to the file. Once a file has some content that content is saved even if the server is rebooted, so data will persist and can be used again. The obvious way to access that data again, and the opposite of `fwrite()` is of course `fread()`. The `fread()` function takes 2 parameters, the first being the handle for the file to be read, and the second being the number of bytes to be read. If it is unknown how many bytes to read in, the `filesize()` function can be used to read in the whole file. In this example `filesize()` will return the number of bytes in the `test.txt` file, and so `fread()` will read the whole file into `$content`.

```
<?php
$file =fopen("test.txt", "r");
$content = fread($file, filesize("test.txt"));
fclose($file);
?>
```

There are alternatives to `fread()`, such as using `fgets()`, which will return a line of text from the open file, stopping at a line break. This can be used alongside `feof()`, which stands for 'file end of file', in a loop that reads in each line until the end of the file. The following code will loop through a file until the end, outputting the contents onto the page.

```
<?php
$file =fopen("test.txt", "r");
while(!feof($file))
{
    echo fgets($file) . "<br>";
}
fclose($file);
?>
```

Just as `fgets()` is related to `fread()`, another function `fputs()` is an alias for `fwrite()`.

```
<?php
$file =fopen("test.txt", "w");
fputs($file, "Hello World");
fclose($file);
?>
```

Another related function called `file_get_contents()` reads the whole contents of a file returning a string, while `file_put_contents()` can be used to write a string to a file. Another way of reading content from a file is to use the `file()` function which reads a file into an array, with each line becoming the next index of the array. There are several other useful PHP functions for managing files.

6.1.1 *basename()*

The `basename()` function returns the filename, given the path to a file. The path to a file contains all the directories and subdirectories necessary to locate a particular file, while `basename()` will simply return the name of the file. This could be useful when a path is stored as a variable.

```
<?php
$path = "/www/code/test/index.php";
echo basename($path); //outputs "index.php"
echo basename($path, ".php"); //outputs "index"
?>
```

6.1.2 *mkdir(), rmdir(), chmod()*

PHP can create, delete and edit directories within the file system, including changing the access permissions to those directories or files within directories. Firstly to create a new directory, or folder, the `mkdir()` function is used.

```
<?php
mkdir("newfolder", 0777);
?>
```

This example creates a new directory called "newfolder". The second parameter sets access permissions to the new directory, but is optional, and by default "0777". Access permissions consist of 4 numbers;

- The first number is always 0
- The second number is for permissions for the owner
- The third number is for the owner's user group
- The fourth number is for everybody else

There are a variety of different values available, depending on whether the user should be able to read, write or execute files stored in the directory.

- 1 = execute permissions
- 2 = write permissions
- 4 = read permissions

Given this example 0777 allows read, write and execute permissions for all users. While this is sometimes the intended security levels, clearly this may cause security issues. Security is a major issue within web development, which is discussed further later, but clearly the access permissions to folders needs to be managed carefully. Common alternatives are;

- 0644 – Read and write for the owner, but only read for anyone else

- 0755 – Everything for the owner, but read and execute for everyone else

The opposite of `mkdir()` is of course to remove a directory, which can be done using `rmdir()`.

```
<?php
rmdir("newfolder");
?>
```

The function for changing access permissions to a file or directory is `chmod()`. `chmod()` takes two parameters the name of the file (or directory), and the new access permissions.

```
<?php
chmod("newfolder", 0644);
?>
```

6.1.3 `scandir()`, `is_dir()`, `is_file()`, `is_executable()`, `file_exists()`

When navigating a file system it may be necessary to find out what files and subdirectories exist within a file, `scandir()` will return an array of the contents of a directory. Any directory may contain a number of files and a number of subdirectories, so PHP has functions for determining which members of the array are files and which are directories; both `is_dir()` and `is_file()` return a Boolean. Some files could be executable, and the `is_executable()` function will test that. Also related to this `file_exists()` can test whether a filename is contained within a directory.

6.1.4 `copy()`

The copy function simply makes a copy of a file, taking two parameters, the source of the file, and the target of the file. The function returns TRUE when it successfully copies a file.

```
<?php
copy("source.txt", "target.txt");
?>
```

6.2 Introducing MySQL and phpmyadmin

Files offer a convenient way for developers to store data on the server so that the data remains even when nobody is connected to the website. The calendar appointments from the last chapter could be written to a file, and then read in when the page is reopened. However, files do have several limitations. Firstly, as files become larger they are slower and less convenient to work with. Searching for particular data within a file can be difficult – consider searching a file with all the appointments created to find all the appointments for a particular day, for a particular user. Reading from files begins at the start of the file, so data is accessed sequentially. When writing to files, the options are to overwrite the file, or append to the end of the file. Randomly reading, writing or deleting in the middle of a file is not straight forward. Managing data stored in files is also limited, there are access permissions, but only with limitations. There is also a potential problem with concurrent access to a file, when perhaps more than one user tries to read or write a file at the same time. For these reasons we will examine an alternative to using files, introducing the database.

MySQL is a database server that scales ideally for both small and large applications. Standard “Structured Query Language” or SQL is used to interface with the database. It is free and easy to use, coming as part of the web development stack discussed previously. In MySQL data is stored in database objects called tables. A table being a collected of related data, consisting of rows and columns, much like a spreadsheet. Data can be related across multiple tables allowing more complex queries to be run. The phpmyadmin tool is very useful for looking at the database, which can be accessed via localhost/phpmyadmin.

The phpmyadmin tool provides a graphical user interface (GUI) for easily manipulating the databases & tables. When first opened, the first page provides some information about which version of the server is being used and any warnings about the configuration, a list of any existing databases and a menu that looks similar to the image below.

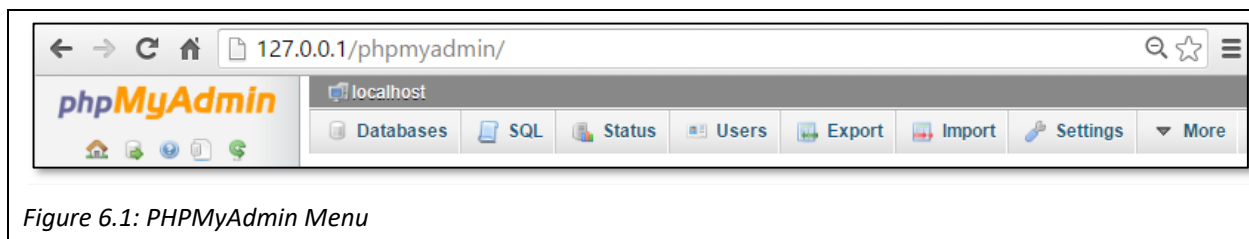


Figure 6.1: PHPMyAdmin Menu

We will soon see how PHP can be used along with SQL to interact with the database, but sometimes tasks can be simpler using phpmyadmin, for example when first creating a database and setting up tables. Simply click on the “Databases” tab, and there is an option to explore an existing database or to create a new one.

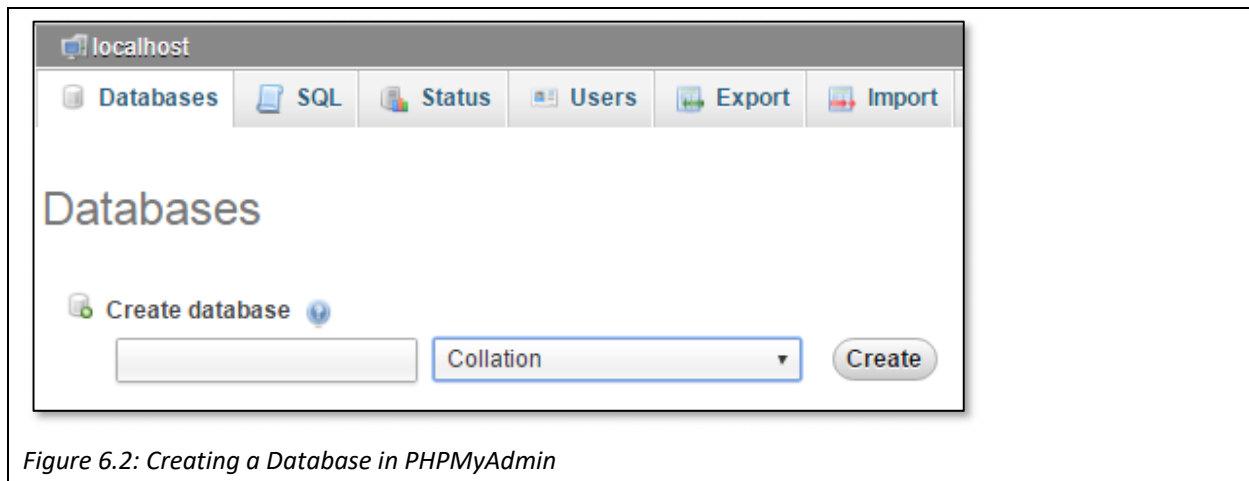


Figure 6.2: Creating a Database in PHPMyAdmin

To create a database, just input a name. For now we can ignore the “Collation” drop down menu. Collation refers to the character set (charset) being used, and how string matching is performed in the database. We will be more concerned with security later, but the “Users” tab can be used to create new users and allocated them different levels of access. By default the root user is used. The “Import” and “Export” tabs should be self-explanatory, and particularly useful for backing up a database.

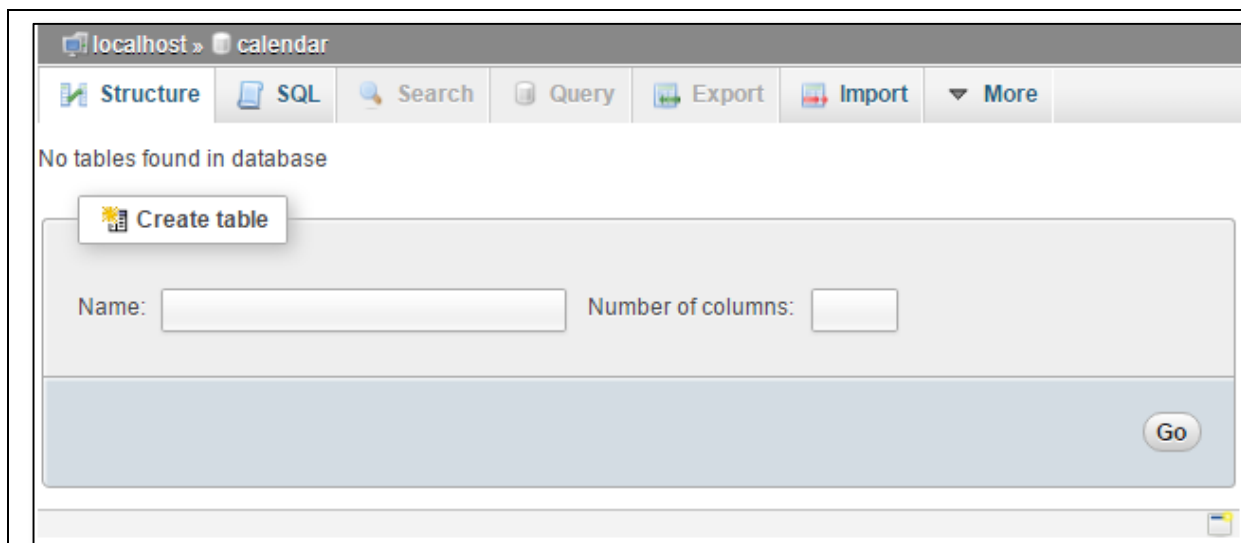


Figure 6.3: Creating a Table in PHPMyAdmin I

Initially when a new database is created, it contains no tables, and no data, but again phpmyadmin provides an intuitive GUI to set up the database’s structure. When a new table is created, it is necessary to specify the name of each column, the type of data it will contain, and often the size of the data. There is a lot more to database design, and normalization, including indexes and foreign keys. This book won’t cover these topics in depth, but recommend interested database designers to investigate further. For the purpose of demonstration, we can use phpmyadmin to create a simple table that can contain data about calendar appointments. It will have 4 columns.

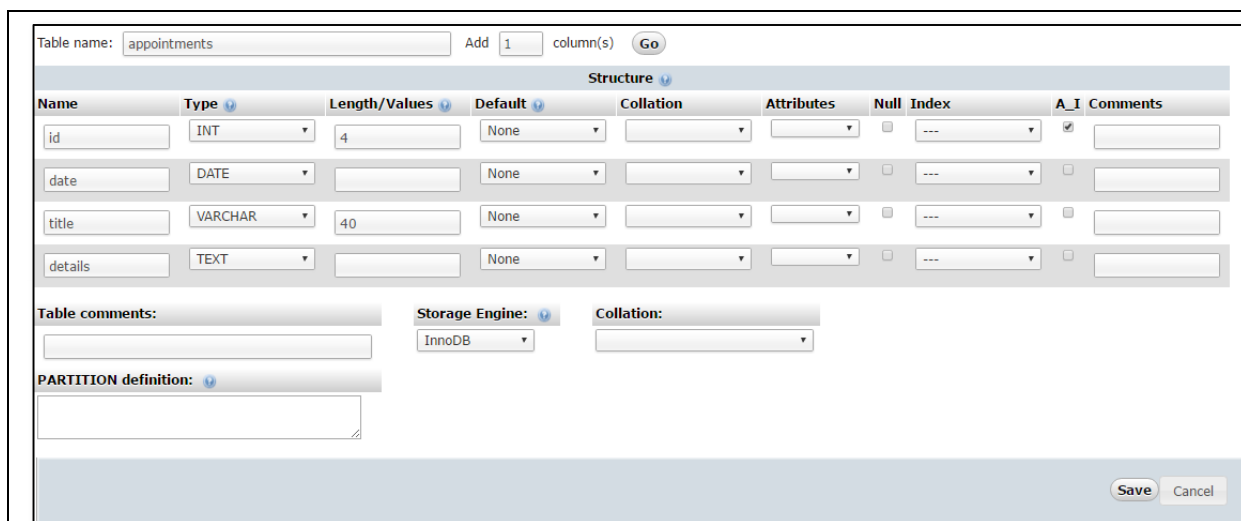


Figure 6.4: Creating a Table in PHPMyAdmin II

In this example, the first field is used to store a unique id for each entry. The type of data is an integer, with a length of 4. The A_I, or auto increment, field is checked, so new appointments will automatically be given the next number. The second field stores a date. The third stores the title, which has the type “VARCHAR”,

or variable character field, which can store letters or numbers, in this case the length of the title is capped at 40. The details of the appointment can be stored in the fourth field, which uses the type text. Once the table is created, phpmyadmin provides different ways to view and interact with it. Notice in the following screen the view is of the appointments table, in the calendar database, stored on localhost.

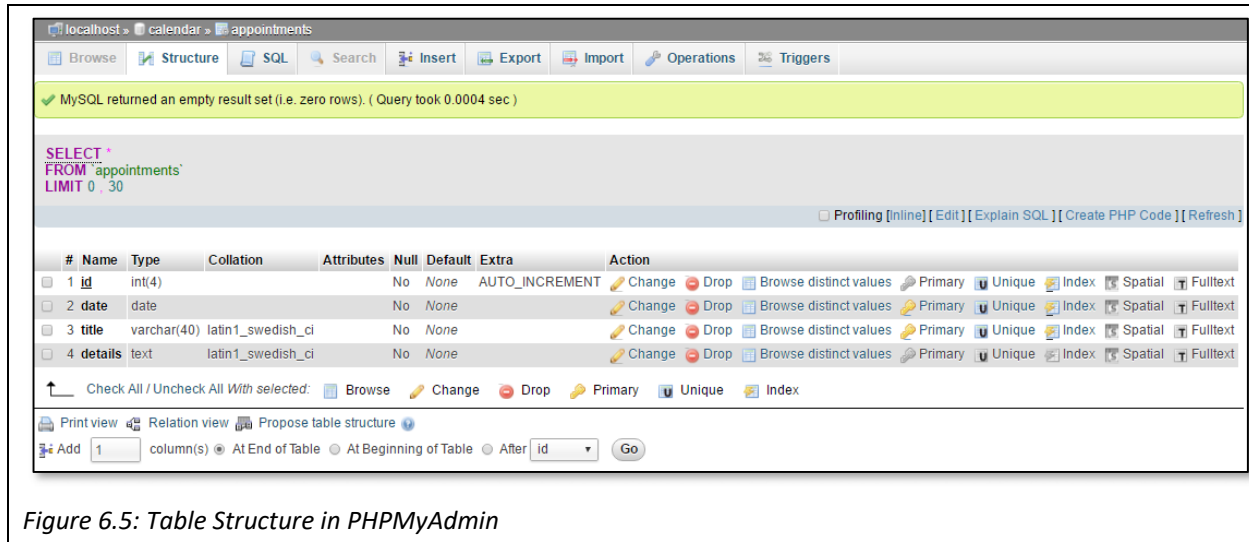


Figure 6.5: Table Structure in PHPMyAdmin

Clearly, as the table has just been created, it is empty, so a MySQL query returns zero rows. There are several tabs available that are worth exploring. The first tab “Browse” allows you to view the data stored in the database - this opens by default, but only when there is data to view. The second tab, “Structure”, is for viewing how the data is organized and structured. This is displayed above and gives metadata about the various columns in the table. The third tab is the “SQL” tab, which allows a database administrator to run SQL queries on the table.

6.3 Structured Query Language

Structured Query Language, or SQL, is a database language for accessing a variety of databases, including MySQL, Oracle, Access and others. The screenshot above shows a first SQL statement, which returned no results.

```
SELECT * FROM appointments LIMIT 0, 30;
```

The general syntax of a SELECT statement is as follows. Each of these examples can be tested using the “SQL” tab in phpmyadmin.

```
SELECT column_name1, column_name2 FROM table_name;
```

The SELECT keyword is used for retrieving data from a database, and the query can specify which columns to retrieve from which table. Notice that using the wildcard “*”, all columns will be retrieved. Also adding the LIMIT keyword, not all results will be displayed. In this case the first 30 results would be returned, or specifically the results between 0 and 30. A WHERE clause can also be used to refine the results returned by a SELECT statement. The following query would only return results where the id equaled 1.

```
SELECT * FROM appointments WHERE id=1;
```

In this case a single '=' sign is used as the operator for testing equality. Other operators may be used in a WHERE clause, including '<>' for not equal, '<', '>', '<=' and '>='. Queries can become more sophisticated using the Boolean operators 'AND' and 'OR'.

6.3.1 INSERT Statement

The "INSERT" Statement is used to add data into a database. The basic structure of an INSERT statement is to specify the table name and then the data to add to it. A new row can be created by adding appropriate values.

```
INSERT INTO table_name VALUES (value1, value2,...);
```

In a database it is common that some values don't exist, and so some fields are left blank, or as NULL. Therefore it is common to specify which columns to insert data into using the following form.

```
INSERT INTO table_name (column1, column2,...) VALUES (value1, value2,...);
```

Using this version we could insert data into the appointments table.

```
INSERT INTO appointments (date, title, details) VALUES ("2016-08-05", "Meeting", "All day in room 401");
```

Now the table can be browsed using the SELECT statement introduced earlier, and the new data will appear. Notice that although no data was inserted into the auto-incremented id column, the row has been given id 1.

The screenshot shows the PHPMYAdmin interface for the 'appointments' table. The top navigation bar includes 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', 'Operations', and 'Triggers'. A status bar indicates 'Showing rows 0 - 0 (~1 total), Query took 0.0003 sec'. The SQL query editor contains the following query:

```
SELECT *  
FROM `appointments`  
LIMIT 0, 30
```

Below the query editor, there are controls for 'Show: Start row: 0', 'Number of rows: 30', and 'Headers every 100 rows'. The table data is displayed as follows:

id	date	title	details
1	2016-08-05	Meeting	All day in room 401

Below the table, there are options for 'Check All / Uncheck All With selected:', 'Change', 'Delete', and 'Export'. The bottom control bar is identical to the one above the table.

Figure 6.6: Browsing a Table in PHPMYAdmin

6.3.2 UPDATE Statement

The “UPDATE” can be used to change data that is already stored in a database. The syntax for an UPDATE statement is as follows.

```
UPDATE table_name SET column1=value1, column2=value2 WHERE column=value;
```

The UPDATE statement includes a “WHERE” clause, which should not be omitted, otherwise all data may be updated. An example UPDATE statement for the previous example could be as follows.

```
UPDATE appointments SET details="Meet at 9AM" WHERE id = 1;
```

Naturally this statement will overwrite the existing contents of the details field.

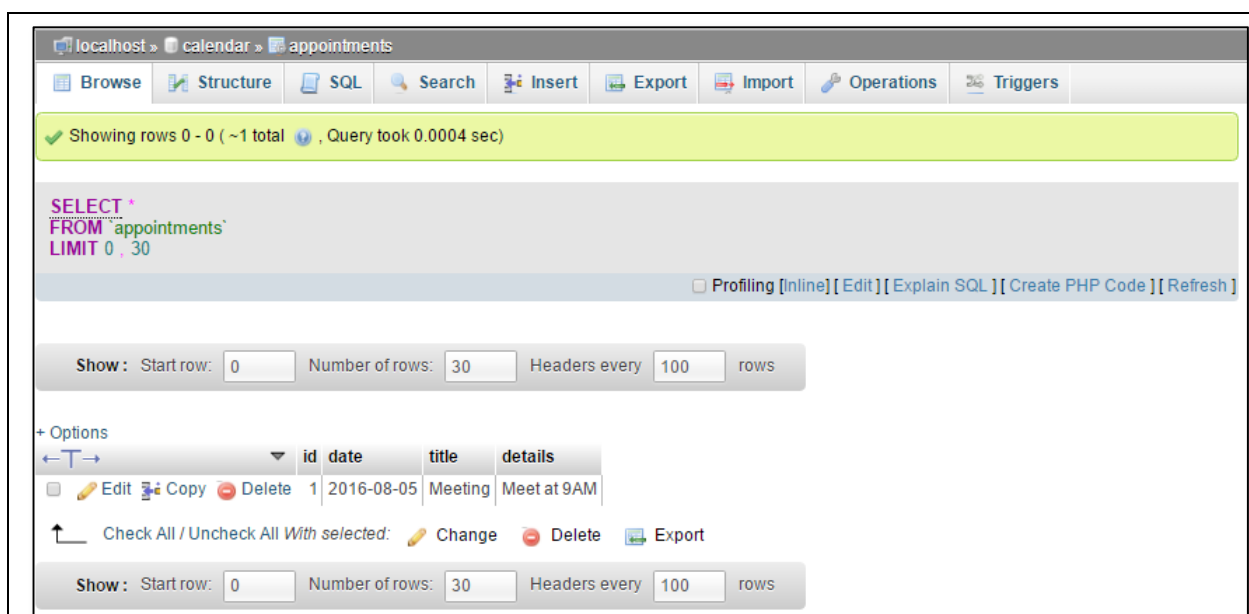


Figure 6.7: An Updated Table in PHPMysqlAdmin

6.3.3 DELETE Statement

The previous statements are useful for writing to a database, updating a database, and reading from a database. A further important operation is to delete from a database, and this uses the “DELETE” keyword. The general syntax for DELETE is.

```
DELETE FROM table_name WHERE column_name = value;
```

Note that the WHERE clause is important, as without it all the data in the database could be deleted. The data in the example table could be deleted using the following command, resulting once more in an empty database.

```
DELETE FROM appointments WHERE date = "2016-08-05";
```


These examples have demonstrated some basic SQL queries – there are plenty of more advanced enhancements that can be done to improve queries, but the basics of inserting, updating, selecting and deleting are enough to get started. These examples can be explored using phpmyadmin, which means that they could be performed by a database administrator, or anyone who has access to the server. When writing web applications it is much more likely that users will want to update data in the database, without needing to understand SQL. As the database is stored on the server we can use PHP to communicate with the database. In the next section we will investigate the API that PHP offers for working with MySQL.

6.4 MySQLi

The MySQLi extension is an API used for manipulating a MySQL database using PHP. It is short for MySQL improved, due to being an improvement on the original MySQL PHP extension. There are alternatives, but this chapter will focus on MySQLi. MySQLi offers 2 alternative means; a procedural approach and an object oriented (OO) approach. This chapter will explore both.

6.4.1 Establishing a connection

The first step in the process of PHP communicating with a database is to establish a connection. This can involve the username and password to ensure the script has permissions to access the data. Once the connection has been established queries can be made on the database, until finally the connection needs to be closed. It is very often a good idea to move the connection scripts to a separate database file, perhaps called 'database.php', and then include this file whenever another file needs to connect to the database. Repeating the database connection script in multiple files can cause security issues as well as creating headaches when passwords need to be changed in the future.

To connect and disconnect using the procedural approach, the following functions are called.

```
$link = mysqli_connect("localhost", "username", "password", "databasename") or  
die("Error" . mysqli_error($link));  
//Some Code  
mysqli_close($link);
```

In this case the script attempts to connect to a database by calling the `mysqli_connect()` function. This function takes 4 parameters, the server (in this case localhost), username and password, and finally the database to connect to. If it fails to connect, an error can be generated. The `mysqli_connect()` function returns a handle which is then used as a reference for any further queries. In this way it is possible for a script to have multiple connections open to multiple different databases. When the database connection is no longer needed it is released by calling the `mysqli_close()` function. This is known as the procedural approach, because it uses procedures or function calls to communicate with the database. The alternative approach is to create a database object to manage the connection. The equivalent OO approach is as follows.

```
$mysqli = new mysqli("localhost", "username", "password", "databasename");  
// Some Code  
$mysqli->close();
```

The code here is similar, however now the `$mysqli` object is created, and member functions for this object can be called using the arrow (`->`) notation.

6.4.2 Querying the database

Once the connection has been successfully established, queries can be called in different ways, depending on whether the procedural or OO approach has been used. It is a good idea to create a string variable for the query and then pass this to a function call.

```
$qry1 = "CREATE DATABASE calendar";  
$qry2 = "CREATE TABLE users ( FirstName varchar(15), LastName varchar(15) )";  
$qry3 = "INSERT INTO appointments (date, title, details) VALUES ('2016-08-06', 'Exam', '9AM')";
```

Notice how different quotes are used to distinguish the string values in the final example. These three examples could be used to create a database, create a table and insert data into the appointments table. Sometimes it is worth testing the query using phpmyadmin before using it in the code as errors are provided directly there while if there is a mistake in the query executed by PHP, there may be no error, but the code may simply not run. Any of these examples can then be executed as follows.

```
mysqli_query($link, $qry1); //procedural approach  
$mysqli->query($qry3); // OO approach
```

Notice that the procedural approach calls the `mysqli_query()` function with 2 parameters, firstly the handle created earlier during connection, and secondly the query. The OO approach calls the `query()` member function for the database object created earlier. This approach can be used for most of the queries discussed so far – the exception being a `SELECT` statement, as PHP needs to manage the results returned from the database after this query.

6.4.3 PHP and `SELECT` statements

The `SELECT` statement will return a number of rows of data depending on how many rows satisfy the select query. These rows of data can then be stored in a variable, an array or an object, that can be further processed by PHP. The first examples only demonstrate finding out how many rows are returned by the query, rather than processing them in any way.

```
$link = mysqli_connect("localhost","username","password","databasename");  
$qry = "SELECT * FROM appointments";  
if ($result = mysqli_query($link, $qry)) {  
    echo "There are " . mysqli_num_rows($result) . " rows!";  
    mysqli_free_result($result);  
}  
mysqli_close($link);
```

This procedural example shows a query which selects every entry from the appointments table. Having called the `mysqli_query()` function, the results are stored in a special kind of variable called `$result`. This variable contains a `mysqli_result` type object, and shortly we will examine how this object can be used. For now, it can

be sent as a parameter to the `mysqli_num_rows()` function which returns how many rows have been found in the database. Notice that once the result is no longer needed it is sent as a parameter to the `mysqli_free_result()` function, which frees up the memory associated with the query. While initially the memory used by a query like this may appear small, it is good practice to tidy up memory after the query. The OO approach is somewhat comparable, but this time using the member functions `num_rows()` and `close()`.

```
$mysqli = new mysqli("localhost", "username", "password", "databasename");
$query = "SELECT * FROM appointments";
if ($result = $mysqli->query($query)) {
    echo "There are " . $result->num_rows . " rows";
    $result->close();
}
$mysqli->close();
```

In both cases the `$result` variable contains the `mysqli_result` object containing the rows of data that satisfied the query. This result data object can be used in several different ways. The first thing to realize is that the data object will contain a number of rows. Each row can be retrieved from the data object in different formats; an array, an associative array, or as an object. Because the number of results is initially unknown, a while loop is a sensible control structure for retrieving data. Lets first consider fetching the results as an array using the `fetch_array()` function.

```
$query = "SELECT * FROM appointments";
$result = $mysqli->query($query);
while($row = $result->fetch_array())
{
    print_r($row);
}
```

The `print_r()` function shows the `$row` array to examine how the data is stored. Assuming there is a single entry in the database, the following might be output.

```
Array ( [0] => 1 [id] => 1 [1] => 2016-08-15 [date] => 2016-08-15 [2] => Meeting [title] => Meeting [3] => Room 401 [details] => Room 401 )
```

Figure 6.8: The Results of `fetch_array()`

The `$row` variable is now an array of the data from the row of the database. Each bit of data is stored twice in the array, both as an indexed array and as an associative array. Therefore the date for the appointment could be referenced as either `$row[1]` or `$row['date']`. Clearly to use the numerical index `[1]` the developer needs to remember the order of columns in the database, while using the associative index `['date']`, the index is taken from the column name. Data can be retrieved from the database by only using the associative array using `fetch_assoc()`, as follows.

```

$qry = "SELECT * FROM appointments";
$result = $mysqli->query($qry);
while($row = $result->fetch_assoc())
{
    print_r($row);
}

```

This time only the associate index array is returned, so the date could be referenced using \$row['date'].

```

Array ( [id] => 1 [date] => 2016-08-15 [title] => Meeting [details] => Room 401 )

```

Figure 6.9: The Results of fetch_assoc()

The final way to return data is as an object. When using the fetch_object() function the variable \$row is returned as an object.

```

$qry = "SELECT * FROM appointments";
$result = $mysqli->query($qry);
while($row = $result->fetch_object())
{
    print_r($row);
}

```

As the returned data is an object, each element is referenced using the arrow notation, such as \$row->date.

```

stdClass Object ( [id] => 1 [date] => 2016-08-15 [title] => Meeting [details] => Room 401 )

```

Figure 6.10: The Results of fetch_object()

6.4.4 Procedural or OO Approach

This section of the chapter has demonstrated that MySQLi has a dual interface, offering both a procedural and OO approach. Readers may wonder why both approaches are discussed and which approach

Try it yourself!

The last chapter concluded by introducing a calendar, but each time the browser was closed, all appointments were lost. This chapter has introduced ways that the calendar can be improved, by storing appointments in a database. A few improvements can be made – firstly when submitting a new appointment form, PHP can catch the post variables and INSERT into the database. Secondly when displaying the calendar page, the database can be queried to return all appointments for that month.

is recommended as best. In reality both approaches work in the same way, and from a performance perspective they are equivalent. The procedural approach is similar to the original MySQL extension before it was replaced by MySQLi, so experienced developers will often find the procedural approach easier to adapt to. However, there is a general trend towards more OO style in PHP, so perhaps there is good reason to adopt the OO approach.

Key Points

- Because PHP is code running on the server, it can be used to read and write data from files stored on the server and there are several functions to help with that.
- There are different modes when opening a file, depending on whether you intend to read, write or append the file.
- Appropriate permissions should be set to limit access to files stored on the server.
- MySQL is a database server that is a key part of the WAMP / LAMP stack.
- Phpmyadmin is a useful tool allowing a GUI for managing databases.
- Structured Query Language or SQL is a database language used to access, query, insert, update and delete from databases.
- MySQLi is an improved PHP extension allowing an API for manipulating a MySQL database via PHP.
- There are choices between a procedural or object-oriented approach to MySQLi.

Further Resources

- 1) w3schools has further information on using PHP to manage files, which can be found here:-
https://www.w3schools.com/php/php_file.asp
- 2) tutorialspoint has a short tutorial on managing files, which can be found here:-
https://www.tutorialspoint.com/php/php_files.htm
- 3) A thorough guide and function reference for the MySQLi extension can be found here:-
<http://php.net/manual/en/intro.mysql.php>
- 4) A further MySQLi function reference guide can be found here:-
https://www.w3schools.com/php/php_ref_mysql.php

Assignment

Last chapter produced a simple calendar where an appointment could be displayed. The main limitations were that when the user left the calendar, the appointment was lost, and only one appointment could be displayed. Improve the calendar from that chapter. When the user creates their appointment, store the information in a database, so that when the calendar is displayed the database is queried and all appointments in the database are then displayed.

Chapter 7

Cookies, Sessions & Security

Objectives

Previous chapters have introduced PHP, a scripting language for running code on the server. This also included code for storing data on the server both in files and databases, which immediately raises security threats. Little was discussed about how to deal with security threats, so this chapter aims to address this issue. One aspect of securing parts of the web involves authentication, which is commonly achieved by users entering their username and password to access certain information. This chapter will discuss creating a login page, and how cookies and sessions can be used to identify users. After reading it you should;

- Understand that HTTP is a stateless protocol, which means that the server by default doesn't maintain the state of each client that makes a request.
- Be able to use cookies to store small amounts of information within the client's browser, and also remove those cookies.
- Understand the similarities and differences between cookies and sessions.
- Be able to create a registration and login system to authenticate visitors to your site and to secure certain pages.
- Understand some of the key security threats such as SQL injections, and how to counter these threats.

Contents

- 7.1 Cookies
- 7.2 Sessions
- 7.3 PHP Login Script
- 7.4 Online Security Threats

The previous chapter introduced how PHP can be used to store data permanently on the server, both in files and in databases. While the code works, and allows developers to read and write from the server, very little consideration was given to security. Clearly when allowing a client computer to manipulate files on the server, there can be security concerns such as SQL injections or brute force attacks, and this chapter will address some of these issues. Web security is a constantly evolving field as hackers find new ways to exploit security flaws, and developers adopt new practices to protect their assets. Therefore this chapter isn't a complete guide to web security, but will raise some of the issues and resolutions.

Considering the calendar example introduced in previous chapters, clearly only the owner of the calendar should be able to add or edit appointments, and depending on the purpose of the calendar, perhaps be the only one to view appointments. The standard way of ensuring that only the right people have access to certain pages is through authentication, most often requiring the user to input a password. Previously this book has discussed form validation, and a simple login form could easily be created using an input field of type 'password'. An issue arises however because Hypertext Transfer Protocol (HTTP) is a stateless protocol.

HTTP is the foundation data communication protocol used throughout the web, crucial for transferring data between clients and servers. HTTP is a stateless protocol, which means that each request is treated independently from any other request, in other words the server doesn't keep track of the state of any client that makes a request from it. This is a benefit for the server, as makes the role simpler, a conversation doesn't need to be remembered and each request is treated individually. Should the client part of a conversation lose connection, the server doesn't use unnecessary memory to store information about the absent client. However, extra data may be needed in each request to authenticate the request.

HTTP is a stateless protocol, which means that each request is treated independently from any other request, in other words the server doesn't keep track of the state of any client that makes a request from it.

This chapter will discuss how this can be resolved as well as introducing some of the security related issues that developers should be aware of.

7.1 Cookies

Cookies are small pieces of data sent from a website and stored in the users browser on the client computer. The intention for cookies was to allow stateful information through HTTP requests. For example to store items the user has in their shopping cart, their name, or also to record their browsing history, such as which products they have viewed etc. A particularly important type of cookie is an authentication cookie, which is used to know whether the user is logged in, and whether the server should send a page containing sensitive information. Because HTTP is a stateless protocol, and doesn't maintain information about each request, it needs a way to verify which requests come from authenticated users, and which from simple visitors.

Much like a variable, a cookie consists of a name value pair. The cookie is originally set by the webserver, and stored on the client computer. The client computer then sends cookie data back to the server

with any request, which allows the client machine to identify itself to the server. Cookies can contain data up to 4,096 bytes in size, and each domain can set at least 50 cookies. There is of course a tradeoff between storing this information on the client machine, versus storing data on the server. With data stored on the client machine, the user is able to see what data is being stored – cookies can be seen in the “Resources” tab of develop tools using a chrome browser. Because cookies have received some negative press due to potential security threats, some users choose to disable cookies and prevent websites from storing data on their machine.

A cookie can be set using the PHP `setcookie()` function, before the opening `<html>` tag when a page is sent to the client. The `setcookie()` function can take several parameters, beginning with the name and value, and also a time for the cookie to expire. In this example, the cookie expiration is set for one day later by using a current timestamp, and adding the number of seconds in the next 24 hours.

```
$name = "user";  
$value = "John";  
setcookie($name, $value, time() + (60*60*24));
```

Other optional parameters for the `setcookie()` function include being able to set the path and domain, limiting where the cookie is accessible from, specifying that the cookie should only be sent over a secure https connection, and an `HTTPOnly` parameter which prevents the cookie from being used by JavaScript. Once a cookie has been set, it is then sent along with any page request to the server within the `$_COOKIE` array. The `$_COOKIE` array is a superglobal array, similar to the `$_POST` and `$_GET` arrays introduced previously. The `isset()` function can be used to check if a cookie exists within the `$_COOKIE` array.

```
if(isset($_COOKIE['user']))  
{  
    $user = $_COOKIE['user'];  
}
```

Cookies will expire according to the expiry timestamp specified when they are created, which means they can be destroyed by specifying a timestamp in the past. The timestamp used though depends on the client machine, so a timestamp a long time in the past might be sensible.

```
setcookie("user", "John", time() -1);
```

Cookies offer developers the means to identify which client has made the HTTP request, and so enables personalized responses; for example having retrieved the user cookie created above, the page could say “Welcome back John”. Beyond this, a profile could be created for each user and alternative pages delivered according to their personal preferences. Cookies can also be used to manage sessions, storing whether a user is logged into a website. As the `$_COOKIE` array is sent along with each request, cookies can also be used to track user behavior across a site.

Cookies, however, do present security and privacy concerns. Some users are concerned about the ability to track their activity using cookies, and will simply disable cookies. Cookies are stored in the browser

on a machine, which doesn't necessarily identify a user, as one user may use multiple browsers, or multiple users may use the same machine.

7.2 Sessions

Cookies store information about a user on the client's machine, sessions are similar to cookies, but the information is stored on the server, with an identifying cookie residing on the client machine. When working with a PHP application, a user might login, perform tasks and then leave – this is essentially a session – a period of time doing a task. As already noted though, the HTTP requests do not maintain the state of each request, using PHP sessions affords another way of tracking multiple requests by a user over a period of time. Data stored in a session will persist as long as the session continues, unlike data stored in a database which will remain permanently.

PHP sessions are created by calling the `session_start()` function, variables can then be stored in the `$_SESSION` super global array, which works in a similar way to the `$_COOKIE` super global array.

```
<?php
session_start();
$_SESSION['name'] = "John";
echo $_SESSION['name'];
?>
```

This code creates a session variable called `name` which stores the value "John". This session variable is available across any subsequent pages that start the session by calling `session_start()`. Essentially when a session is opened a user key is stored as a cookie on the client machine, this cookie is a random string of 32 hexadecimal digits, such as "fca17f071bbg9bf7f85ca281653499a4". When another page calls `session_start()` the cookies are scanned for an appropriate user key which allows the server to retrieve the appropriate `$_SESSION` array.

Session variables can be removed by calling `session_unset()`, and a session can be completely destroyed by calling `session_destroy()`. The `php.ini` file can be edited to control how long a session will last, but by default session variables will generally last for 24 minutes. Server side cookies can store large amounts of data, much more than the limited size of client side cookies. Also with the data being stored on the server it can't be edited by the user.

7.3 PHP Login Script

This section will demonstrate using sessions to create a registration and login script, which will allow pages to be hidden and only accessible to authenticated users. Whilst it may not be completely hacker proof, it serves to highlight some of the security threats that need to be considered.

7.3.1 Setting up the Database

To begin with a database needs to be created, which can be done using `phpmyadmin`.

```
CREATE DATABASE 'phplogin';
```

To make the database more secure, we won't access it using the root admin access, instead create a new user which only has SELECT, UPDATE and INSERT privileges. The new user will be used to connect to the database without the ability to DELETE or DROP data. If you did need to delete data using PHP for some reason, you could create a separate user with delete privilege. This means that if the script was somehow hacked, at least the hacker wouldn't be able to delete the data. This can be done using the GUI interface in phpmyadmin, or by the following SQL statements. In this case the user 'boss' is created, with the password 'Bos\$123', although clearly you may wish to generate a more secure password.

```
CREATE USER 'boss'@'localhost' IDENTIFIED BY 'Bos$123';
GRANT SELECT, INSERT, UPDATE ON phplogin.* TO 'boss'@'localhost';
```

The database will contain a table that stores the users information, with their username, email address and password. The table can be created using phpmyadmin, or with straight SQL.

```
CREATE TABLE 'phplogin'. 'users' (
  'id' INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  'username' VARCHAR(30) NOT NULL,
  'email' VARCHAR(50) NOT NULL,
  'password' CHAR(128) NOT NULL
) ENGINE = InnoDB;
```

7.3.2 Connecting to the Database

The next step is to allow PHP to connect to the database, as discussed in the previous chapter. The database connection script will be separated from each page and just needs to be included when connection to the database is needed, therefore these files can be stored in an 'includes' folder within the project directory.

```
<?php
define("HOST", "localhost");
define("USER", "boss");
define("PASSWORD", "Bos$123");
define("DATABASE", "phplogin");
?>
```

The first file contains the login credentials, which on a live server would ideally be stored outside the document root so that it couldn't be accessed by chance via a URL. This file simply defines the login credentials set up previously and is saved as '/includes/db-config.php'. This credentials config file means the username and password are stored once in a single file, which makes it easier to update when passwords need to be changed. A second file can be used to create a mysqli object by including the credentials defined in db-config. This file is called '/includes/db-connect.php'.

```
<?php
include_once 'db-config.php';
$mysqli = new mysqli(HOST, USER, PASSWORD, DATABASE);
?>
```

This db-connect file can then be included wherever database connections are needed, making the mysqli object available, whenever the database needs to be accessed.

7.3.3 Session Management

As described in section 8.2, the session_start() function needs to be called to initiate a session and store data, such as login data, within session variables. There are ways, not discussed here, in which the session_start() function could be made more secure, so it makes sense to move the session function to a separate file that can be included whenever the session information is needed. The separate file can also include some session related functions that can be useful for checking if a user has logged in. The following is saved in the sessions.php file within the 'includes' folder.

```
<?php
session_start();
function logged_in() {
    return isset($_SESSION['username']);
}
function confirm_logged_in() {
    if (!logged_in()) {?>
        <script type="text/javascript">
            window.location = "login.php";
        </script>
        <?php
        }
    }
?>
```

In future files the sessions.php file can be included to initiate a session. There are two further functions included in the file. First the "logged_in()" function returns true or false, depending on whether a username has been set in the \$_SESSION array. The second function "confirm_logged_in()" will redirect the browser to the login page, if the logged_in() function returns false.

7.3.4 Logging In

The login script consists of several pages, but begins with a form for the user to input their username and password. This is contained in the login.php file.

```

<?php include_once('includes/session.php');?>
<html>
  <head>
    <script type="text/JavaScript" src="js/sha512.js"></script>
    <script type="text/JavaScript" src="js/functions.js"></script>
  </head>
  <?php if (logged_in()) { ?>
    <script type="text/javascript">
      window.location = "member.php";
    </script>
  <?php } ?>
<body>
  <h4>Log In</h4>
  <form action="includes/processlogin.php" method="post">
    Username: <input name="username" type="text" autofocus><br>
    Password: <input name="password" type="password" id="password" value=""><br>
    <input type="submit" name="login" value="Login" onclick="formhash(this.form);">
  </form>
  Not registered? <a href="register.php">Register Here</a>
</body>
</html>
?>

```

Obviously this form could be styled using CSS to make it more attractive, and also JavaScript should be used to validate the user's entry. As these tasks have been discussed previously, they are removed from this example, but there remains a few interesting complications. Firstly PHP is used to check if the user is already logged in – if they are they are redirected to a page called "member.php".

Two JavaScript files are also included, both contained in a "js" folder. The first is "sha512.js", which can be downloaded from <http://pajhome.org.uk/crypt/md5/sha512.html>. SHA stands for "Secure Hash Algorithm", and is used in this case to create a hash value for the password, so that the password isn't sent from the client computer to the server in plain text. The second JavaScript file, called "functions.js" contains a function to use the hash function to encrypt the password before it is sent through the web. First the getElementById() function is aliased to make life easier referring to it instead by \$. Secondly a formhash() function is defined, notice that this function is called by the "onclick" event for the forms submit button. This function will become clearer in later chapters after studying jQuery. When the form is submitted the function is called and creates a new input on the form, which is hidden. This field, called "p" is then given the hashed valued of the password (after calling hex_sha512). The password field is then cleared so the password will not be transmitted in plain text. This function can be improved to validate the entries for all the user inputs.

```

function $(id)
{
    return document.getElementById(id)
}
function formhash(form) {
    var p = document.createElement("input");
    form.appendChild(p);
    p.name = "p";
    p.type = "hidden";
    var password = $('password').value;
    p.value = hex_sha512(password);
    $('password').value = "";
}

```

The action parameter for the login page transfers to a “processlogin.php” file within the includes directory.

```

<?php
include_once 'db_connect.php';
include_once 'functions.php';
include_once 'session.php';
if (isset($_POST['username'], $_POST['p'])) {
    $username = $_POST['username'];
    $password = $_POST['p'];
    if (login($username, $password, $mysqli) == true) {
        header('Location: ../member.php');
    } else {
        header('Location: ../login.php');
    }
} else {
    echo 'Oops - Please Try Again!';
}
?>

```

If no values for username and password have been sent, then an error message is displayed, otherwise the user is redirected, depending on the result of a function call to the function login(). If the function returns true, the user is redirected to the member.php page (which could be any page containing secured information), while if it returns false, the user is redirected to the login page. The login() function is contained within a php functions page, also in the includes directory.

```

<?php
function login($username, $password, $mysqli) {
    if($stmt = $mysqli->prepare("SELECT id, username, password FROM users WHERE username = ? LIMIT 1")) {
        $stmt->bind_param('s', $username);
        $stmt->execute();
        $stmt->store_result();
        $stmt->bind_result($user_id, $username, $db_password);
        $stmt->fetch();
        if($stmt->num_rows == 1) {
            if ($password == $db_password) {
                $user_id = preg_replace("/^[0-9]+/", "", $user_id);
                $_SESSION['user_id'] = $user_id;
                $username = preg_replace("/^[a-zA-Z0-9_-]+/", "", $username);
                $_SESSION['username'] = $username;
                $_SESSION['login_string'] = hash('sha512', $db_password);
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
}
?>

```

The login function uses prepared mysqli statements, rather than the statements we have introduced previously. With prepared statements the query and the data are sent to the server separately which can assist to protect against an SQL injection. First the query is sent, (or prepared), before the data follows using the bind_param() function, in this case the data is the username, which is a string (or 's'). Assuming that the database contains the username, the query returns the encrypted version of the password as stored on the server. This is then compared with the password given by the user. If the passwords match, session variables are created and the function confirms the user has logged in.

A common attack for this kind of set up would be a brute force attack, where the user could randomly guess passwords repeatedly until the user gets lucky and guesses the correct password. There are several ways to combat brute force attacks, for example locking a username after a specified number of guesses over a certain time period. Currently the best way to manage brute force attacks is to use CAPTCHA (**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part). Using existing CAPTCHA solutions will ensure that a human is attempting to guess the password each time.

7.3.5 Registration

The login script assumes that there are registered users in the database who's passwords can be compared. Therefore to make a login system, a registration page is needed.

```
<html>
<head>
  <script type="text/JavaScript" src="js/sha512.js"></script>
  <script type="text/JavaScript" src="js/functions.js"></script>
</head>
<body>
<h4>Register</h4>
<form action="includes/processreg.php" method="post"><br>
  Username: <input type="text" name="username"><br>
  Email: <input type="text" name="email"><br>
  Password: <input type="password" name="password" id="password"><br>
  Confirm password: <input type="password" name="pass2" id="pass2"><br>
  <input type="submit" value="Register" onclick="reghash(this.form);" />
</form>
<p>Go back to <a href="login.php">login</a>.</p>
</body>
</html>
```

The registration page here consists of a relatively straightforward form asking for the users details. As previously the "sha512.js" function is used to encrypt the password inputted. The onclick event, when the user registers calls another function "reghash()" which is contained in the "functions.js" file. This function is similar to the hash function used in the login form previously, with the main objective being to hash the password to prevent the password being transmitted in plain text. This function should be improved by thoroughly validating all the inputs the user submits.

```
function reghash(form) {
  var p = document.createElement("input");
  form.appendChild(p);
  p.name = "p";
  p.type = "hidden";
  var password = $('password').value;
  p.value = hex_sha512(password);
  $('password').value = "";
  $('pass2').value="";
}
```

The action parameter for the register form is to post the values to the "processreg.php" page. This page is responsible for finally validating the user's input before adding the data into the database.

```

<?php
include_once 'db_connect.php';
$username = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_STRING);
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
$password = filter_input(INPUT_POST, 'p', FILTER_SANITIZE_STRING);
if ($insert_stmt = $mysqli->prepare("INSERT INTO users (username, email, password) VALUES (?, ?, ?)") {
    $insert_stmt->bind_param('sss', $username, $email, $password);
    $insert_stmt->execute();
    header('Location: ../login.php');
}
else
{
    header('Location: ./register.php');
}
?>

```

Once again, prepared statements are used to separate the data from the query, and once again this example is simplified to remove the validation of user inputs. This validation is particularly important, and the validation should be handled both on the client machine using JavaScript and on the server using PHP.

7.3.5 Secured Pages

Once users are logged in they should be able to access secured pages, such as the “member.php” page referenced previously. Any page that should be secured needs to check if the session variables have been set before displaying the page, and if not, reroute the user to a login page. The login_check() function defined earlier can be used.

```

<?php
include_once 'includes/db_connect.php';
include_once 'includes/functions.php';
include_once 'includes/session.php';
?>
<?php
if (login_check($mysqli) == true) : ?>
    <p>Welcome <?php echo $_SESSION['username']; ?></p>
    <a href="logout.php">Logout Here</a>
<?php else : ?>
    <p>You are not authorized to access this page. Please <a href="login.php">login</a>.</p>
<?php endif; ?>

```


7.3.6 Logging Out

The final step to a login system is to allow users to logout, which involves removing all the session variables and finally destroying the session.

```
<?php
include_once 'includes/functions.php';
include_once 'includes/session.php';
$_SESSION = array();
$params = session_get_cookie_params();
setcookie(session_name(), "", time() - 42000, $params["path"], $params["domain"], $params["secure"],
$params["httponly"]);
session_destroy();
header('Location: login.php');
?>
```

7.4 Online Security Threats

The login script described in this chapter is not flawless, but it does attempt to address some of the threats that web developers need to be aware of.

7.4.1 SQL Injections

An SQL injection is a hacking technique where SQL statements are 'injected' into some data entry field. These statements could be used to bypass security and access data that shouldn't be accessed, or delete data, amongst other things. Consider the following query being prepared in PHP where the \$username parameter will come from a text box that the user will fill in.

```
<?php
$sql = "SELECT * FROM users WHERE username = " . $username . " ";
?>
```

On initial inspection the statement makes logical sense, with the developer expecting the user to input their name and the query to return data about that particular user. Now consider if the user inputs the following;

```
' OR '1'='1'; DROP TABLE users;
```

Once PHP parses the input, the full SQL statement may look like this;

```
SELECT * FROM users WHERE username = ' OR '1'='1'; DROP TABLE users;
```

The first query in the statement could potentially return all the data in the users table, while the second query could potentially delete all the data in the table. As illustrated in the code in this chapter, one way of mitigating against SQL injections is to use prepared statements, where user input isn't embedded directly into a statement, instead it is bound to a placeholder, and only accepted if it is of the correct type. An

alternative, but more error prone, approach is to “escape” a SQL statement before sending it by using the PHP function `mysqli_real_escape_string()`.

7.4.2 Session Hijacking

Session hijacking occurs when a hacker manages to steal the session token, perhaps by sniffing the data being sent over a network, or by tricking a user into following a link to a page crafted to use JavaScript to send the token to the attacker, or even by guessing a predictable session token. Once an attacker has a session ID, they are able to pretend to the server that they are the legitimate user. A classic example of this is called the “man in the middle” attack, where an attacker sets up their machine inline between the user and the server, able to intercept and further traffic that passes between the two. A further example is Cross-site scripting, where the attackers tricks the user’s computer into running particular code by making it appear to come from a trusted server.

Ways to avoid session hijacking include making sure that all data traffic is encrypted, perhaps using SSL. Using a long, random string as the session key can make it harder to guess, while some services change the value of the session cookie each time a request is made, reducing the time frame for an attack.

Key Points

- HTTP is a stateless protocol, so the server doesn’t store the state of each client that connects to it.
- Cookies can be used to store data in the user’s browser so that the server can identify which browser has made the request. However, as they are stored on the client machine, the server has limited control over them.
- A session stores data on the server, with an identifying cookie on the client machine.
- Sessions can be used to secure certain webpages, meaning a user has to login to access it.
- Passwords should be secured before being sent over the web.
- SQL injections and session hijacking are just 2 security threats that users need to be protected from.

Further Resources

- 1) The w3schools introduction to cookies can be found here:-
https://www.w3schools.com/php/php_cookies.asp
- 2) Further information about sessions can be found in the php.net documentation:-
<http://php.net/manual/en/book.session.php>
- 3) An alternative guide to creating a secure login script can be found here:-
<https://www.wikihow.com/Create-a-Secure-Login-Script-in-PHP-and-MySQL>

With the source files available here:-

<https://github.com/peredurabefrog/phpSecureLogin>

Assignment

The assignment in previous chapters has been to create a calendar, but thus far the calendar has been publically visible. Write a registration and login script which allows multiple users to keep their appointments securely.

Part 3

Client-Side Scripting with JavaScript and JQuery

Chapter 8

Introducing JavaScript

Chapter 10

JQuery and the DOM

Chapter 9

Introducing JQuery

Chapter 11

Asynchronous JavaScript and JQuery UI

Part three focuses on the JavaScript programming language, which is often considered one of the three core web technologies, alongside HTML and CSS. Whilst the previous part focused on code running on a server, this part focuses on code running on the client machine, beginning with raw JavaScript. JQuery is a library written in JavaScript, which facilitates some cool features such as handling browser events, and enabling some attractive effects. JQuery also helps manipulate the DOM (or Document Object Model). This part also discusses Asynchronous JavaScript allowing parts of the webpage to be loaded independently from other parts.

Chapter 8

Introducing JavaScript

Objectives

This chapter introduces JavaScript, one of the key programming languages used on websites. JavaScript is the main client side scripting language used when running code in all modern browsers. This chapter introduces basic programming concepts and how they are implemented in JavaScript. Readers who have experience using other programming languages should find it easy to transfer their existing coding skills to JavaScript. After reading this chapter you should:

- Know how to run JavaScript on a webpage.
- Be able to use core programming fundamentals in JavaScript, including Variables, Operators, Control Statements, Arrays and Functions.
- Understand how the `getElementById()` function can be used to access and change elements on the page.
- Be introduced to the Document Object Model or DOM.
- Understand how forms can be used to move data from one page to another and be able to use JavaScript to validate the inputs to a form.

It is worth mentioning at this point that this chapter is intended as an introduction to JavaScript, later chapters will explore the JQuery library which is built using JavaScript.

Contents

- 8.1 Variables
- 8.2 Operators
- 8.3 Control Statements
- 8.4 Arrays
- 8.5 Functions
- 8.6 `getElementById()`
- 8.7 Form Validation Example

JavaScript is one of the most important web programming languages, essential for all web developers to understand. While HTML defines the content and structure of a webpage, and CSS is used to define the layout and design of a page, JavaScript is used to bring the page to life. JavaScript is a client side scripting language which runs code within the browser, and has access to all elements in the document. JavaScript is a powerful tool for webmasters, and there are many frameworks and libraries that can be used to make programming in JavaScript easier. In later chapters we will investigate JQuery, a JavaScript library, but the purpose of this chapter is to introduce the basics of JavaScript. Amongst other purposes JavaScript can be used to move elements around on a page, update information on the page, or respond to user interactions such as when the user clicks on an element or rolls over it.

JavaScript is a client side scripting language that is used to bring a webpage to life, running code within the browser and having access to all elements in the document.

Let's dive straight in with a simple "Hello World" example;

```
<script type="text/javascript">
  document.write("Hello World")
</script>
```

JavaScript can be written within script tags that have the type "text/javascript". In this example, the "write" method is called to write the string "Hello World" onto the document, or page. Programmers with experience in other programming languages may recognize aspects of this code, such as the dot notation, and the brackets. Without experience in other programming languages, pasting the above inside the body tags will simply make "Hello World" appear on the page. JavaScript can be placed within either the head or body of a webpage, or included in a separate file. In many cases it is a good idea to place scripts at the end of the body section, so they are loaded after all the elements on the page, as this can speed up page load times. Alternatively, it is a good idea to separate JavaScript into a different file, in much the same way a css file separates the style from the content, a separate JavaScript file can separate the behavior of the page. In this case placing the following in the head of a page would load the script.js file from within the js folder.

```
<script src="js/script.js"></script>
```

Within the script tags code can be written to allow the page to react and change in different ways depending on how the user interacts with it. The user's computer downloads the JavaScript code and then executes it on their machine, regardless of where the server is located. This means the page can then react in different ways to events such as when the user clicks on an element on the page, or any other events within the client's browser. It is important to remember that JavaScript runs on the client's machine, so if we consider the next example, which will display the current date, the date displayed will depend on where the client's machine is in the world, and so will display a different time and date for users in the USA, Europe, or Thailand.

```
<script type="text/javascript">
  document.write(Date())
</script>
```

In this next section we look at some programming fundamentals in JavaScript. For those who've coded before in other programming languages, we examine differences in syntax and style, while for anyone new to programming this is a crash course in coding logic.

8.1 Variables

Variables are used to store data that can be used in the code. In JavaScript variables contain a value, and have a name that is used to reference them. A variable can contain any type of data including integers, floating point numbers or strings of characters. Variables can also refer to arrays or objects. As JavaScript is a weakly typed programming language, the type of data stored in the variable doesn't need to be specified. A name for the variable does need to be declared though. Variable names can be any alpha-numeric combination, and can include the underscore character, in other words; a-z, 0-9 and `_`. A variable name however may not begin with a number. Variable names are case sensitive, so using capital letters or small letters does make a difference. Choosing an appropriate name for a variable makes a programmers life easier. Variables in JavaScript are declared using the "var" keyword, and can have a value assigned to them using the "=" assignment operator.

```
<script type="text/javascript">
  var a = 5;
  var b = 6;
  var c = a + b;
  console.log(c);
</script>
```

This example created 3 variables, a, b and c, assigning the first two the values 5 and 6 respectively, before assigning c as the sum of the first two. Finally the result is logged within the console. Logging values in the console is a very useful debugging tool, as often when JavaScript code doesn't work, no error is generated, just nothing happens. There are a few more interesting points to note regarding this example. Firstly notice that each line ends with a semi-colon – a semi colon is used to separate statements, but in JavaScript a new line can be used instead. The semi-colon is necessary if you want to place multiple statements on the same line. The code will also appear to work in the same way if the variables are not declared explicitly using the 'var' keyword. In other words, the previous code example will work the same as the following one.

```
<script type="text/javascript">
  a = 5
  b = 6
  c = a + b
  console.log(c)
</script>
```

While both versions appear to work in the same way, there is a difference, in that the “var” keyword declares the variable within a set scope, perhaps within the function it is declared. Without the “var” keyword the variable is essentially global, and may encounter another variable with the same name. Initially this is unlikely to cause problems, but may do in larger, more complicated projects.

Quick Tip

The console is a very important tool for developers using JavaScript. Outputting values to the console is very useful for debugging JavaScript code. The console can be accessed through the developer tools in most browsers.

Strings are a type of JavaScript variable used to store text – a sequence of alphanumeric characters or symbols. The text in a string is surrounded by quotes, either single (') or double (") quotes. One string can be assigned to another string using the assignment operator (=). Strings can also be concatenated using the addition operator (+) as the following example demonstrates.

```
<script type="text/javascript">
  var firstname = "John";
  var surname = "Smith";
  var fullname = firstname + surname;
  var name = fullname;
  console.log(fullname);
</script>
```

As already mentioned a string is surrounded by quotes, either single (') or double ("). Sometimes the string needs to contain a quote, and this can be achieved in a couple of ways. Firstly if the string needs to contain a single quote it is simple enough to surround it with double quotes – so long as the opening and closing quotes match there is little difference between using single or double quotes. Alternatively the escape character (\) can be used as shown in the next example. While there is little difference between single and double quotes, there may be advantages to using double quotes, particularly when passing data as JSON (JavaScript Object Notation). Being consistent is the most important part though.

```
<script type="text/javascript">
  var str1 = "I'm John";
  var str2 = 'I\'m John';
</script>
```

The escape character can also be used to insert a new line into a string, by using “\n”.

Quick Tip

Comments! As your code gets longer it becomes more important to add comments. Comments are ignored by the computer, but are there for you and other developers looking at your code, to make it easier to remember how something works. A single line comment can be added using “//”, and over multiple lines using “/*.....*/”.

```
// Single Line comment
/* Comment over
multiple lines */
```

8.2 Operators

We have already encountered some operators, but it is useful to introduce all the types of operator available in JavaScript. Firstly the arithmetic operators.

Arithmetic Operators	Purpose	Notes
+	Addition	
-	Subtraction	
*	Multiplication	
/	Division	
%	Modulus Returns the remainder from a division	
++	Increment (add one to the value)	Unary operator
--	Decrement (minus one from the value)	Unary operator

Table 8.1: Arithmetic Operators in JavaScript

Note that the increment and decrement operators are unary operators, which means they operate on one variable, whereas the other arithmetic operators operate on two values. Secondly the assignment operators. In each case the value from the expression to the right is assigned to the variable on the left.

Assignment Operators	Example	Equivalent
=	a = b	
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Table 8.2: Assignment Operators in JavaScript

JavaScript also has a selection of logical, or Boolean operators.

Logical Operators	Purpose
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
===	Equal to, both in terms of value and type
!==	Not equal to, both in terms of value and type
&&	AND
	OR
!	Not

Table 8.3: Logical Operators in JavaScript

8.3 Control Statements

JavaScript, like other programming languages has a selection of control statements to help with both branching and looping. Branching enables your code to do different things at different times, rather than always doing the same. Looping, or iteration, simplifies code by allowing the same instruction to be performed multiple times. The simplest form of branching is the “if” statement, which combines with the “else if” and “else” statements as in the following example. The if statement begins with a Boolean expression which will result in either “true” or “false”, allowing different blocks of code to be executed in each case.

```
<script type="text/javascript">
  if (a > 100)
  {
    document.write("a is greater than 100");
  }
  else if (a<100)
  {
    document.write("a is less than 100");
  }
  else
  {
    document.write("a is 100");
  }
</script>
```

If the block of code is short, i.e. a single line, then some programmers prefer to use the ternary operator (?) instead of the if statement. The ternary operator also begins with the Boolean expression, but then separates the code for “true” and “false” using a colon, “:”. The main benefit of the ternary operator is that the resulting code is concise, needing a single line of code where the alternative would run over several lines.

```
<script type="text/javascript">
  document.write(a<5 ? "a is less than 5" : "a is not less than 5");
</script>
```

Another alternative that allows the code to branch is using a “switch” statement. While an if statement, and the ternary operator work using a Boolean expression allowing 2 alternatives, the switch statement works using the value of a variable. This allows more than 2 alternatives, in much the same way that nested if statements, or repeated “else if” statements work. The switch statement can be useful for controlling menus. In the following example, the resulting output depends on the value of the variable “choice”, with code supplied for 2 alternative cases and a default case.

```
<script type="text/javascript">
  switch(choice)
  {
    case 1:
      document.write("1 Selected");
      break;
    case 2:
      document.write("2 Selected");
      break;
    case default:
      document.write("None Selected");
      break;
  }
</script>
```

Another collection of control statements support iteration, where the same code is repeated multiple times. If the programmer knows how many times it should be repeated, then a “For Loop” is an appropriate choice. A for loop begins with 3 statements, an initialization, a Boolean test, and an update action. The initialization is used to create a test variable, and then each time the code is looped, the Boolean test decides whether to repeat the loop again, and the update action is executed.

if(initialization; Boolean test; update action)

In the following example, the block of code will execute 5 times, each time displaying the 5 times table. Note how the + operator is used to concatenate the string.

```
<script type="text/javascript">
  for(count = 1; count <=5; count++)
  {
    document.write(count + " times 5 is " + count*5 + "<br>");
  }
</script>
```

Very often it isn't known how many times some code should iterate, it should simply continue while some condition is true. In this case a "while" loop can be used, which will loop until a Boolean expression is true. Care should be taken to avoid infinite loops where the condition is never satisfied.

```
<script type="text/javascript">
  var count = 0;
  while (count < 5)
  {
    document.write(count + " times 5 is " + count*5 + "<br>");
    count++;
  }
</script>
```

There are two types of while loop. The example above is the standard while loop, where the Boolean expression is tested before the code is executed. This means that it is possible that the code never runs if the condition is already satisfied. The alternative "Do...While" loop, demonstrated below, will execute at least once as the Boolean expression is tested at the end of the block of code.

```
<script type="text/javascript">
  var count = 0;
  do
  {
    document.write(count + " times 5 is " + count*5 + "<br>");
    count++;
  } while (count < 5)
</script>
```

8.4 Arrays

An array is a special kind of variable, which can store more than one value. Grouping related data into an array can make managing data complexity much simpler. Consider a scenario where you needed to store grade information for 3 students; here we could simply create 3 variables, perhaps called student1, student2 and student3. It would be quite straightforward to assign a value to each of these variables, and so some basic processing of these 3 values – for example, finding the average grade, the top and bottom grade, etc. However, consider a new subject where there could be 300 students, or more! Managing 300 variables is much more complicated, and this is where an array becomes useful. In this example a simple array is created to store 3 students names.

```
<script type="text/javascript">
  var students = ["John", "Peter", "Mark"];
  document.write(students[0]);
</script>
```

The square bracket syntax indicates that students is an array variable. In this case each student has an index number, and by default JavaScript begins indexing at 0, making that the index for John, while Peter and Mark are index 2 and 3 respectively. The second line of code then refers to the data stored in index position 0, and so writes the string "John" onto the document. Combining an array with a "For Loop" means it is very easy to go through each element in an array and perform some task on it, such as identifying the top score, as demonstrated in the following example.

```
<script type="text/javascript">
var grades = [50, 80, 70];
var topscore = 0;
for(var i=0; i<3; i++)
{
  if(grades[i]>topscore)
  {
    topscore = grades[i];
  }
}
document.write(topscore);
</script>
```

An array can store any type of data, and can have more than 1 dimension. The following example creates a two-dimensional array to link a student's name with their grade. This time to refer to elements in the array two indexes are needed one for each dimension.

```
<script type="text/javascript">
var students = [["John", 50], ["Peter", 80], ["Mark", 70]];
for(var i=0; i<3; i++)
{
  document.write(students[i][0] + " got " + students[i][1] + "<br>");
}
</script>
```

JavaScript also supports associative arrays, where the index doesn't need to be a number. The following example creates an array of 3 countries, where the index is the 2 character country code, and the value is the name of the country. The example also demonstrates another way of using a for loop with arrays.

```
<script type="text/javascript">
countries = {"uk": "United Kingdom", "th": "Thailand", "us": "United States"}
for (country in countries)
  document.write(country + " = " + countries[country] + "<br>")
</script>
```

In several examples we have called the write method (or function), which is a member of the document. The document also contains a links array, which is an array of all the links in the document, indexed starting from 0, in the order those links appear on the page. We could use this array to refer to the address that the first link on the page is linking to as follows;

```
document.links[0].href;
```

Clearly we may not know how many links appear on the page, just as we may not know how many elements are in any array. Each array also contains a member called “length”, which stores the size of the array, so the number of links on a page could be checked as follows;

```
var numlinks = document.links.length;
```

This may become clearer after further discussions about functions and the Document Object Model.

8.5 Functions

As you write more code, it becomes more complex and harder to manage and one important way of organizing code is to use functions. Indeed, one of the keys to becoming a good programmer is finding the right ways to break a large problem down into smaller, more manageable ones. Writing functions means that you can reuse code – writing code once and then using it whenever needed, with different inputs producing different results. JavaScript also comes with many existing functions which can be used to make coding a lot simpler, as we will explore in much more detail later in the book when we investigate the JQuery library. We have already seen the write() method, or function, which we have used to display something onto the document. First, let’s investigate how to define functions in javascript, which has the following syntax.

```
function name(parameter list)
{
  //code
}
```

An example of a simple function that calculates the average of 3 numbers is given below. The function keyword is used to declare that it is a function, followed by the name of the function, in this case “average”. The parameters are specified inside normal brackets, named n1, n2 and n3. The function returns the result of a small calculation. A function is then called by referring to the name and adding arguments for each parameter.

```
<script type="text/javascript">
function average(n1, n2, n3)
{
  return (n1+n2+n3)/3;
}
document.write(average(5,6,7));
</script>
```

Parameters in JavaScript are 'call by value' parameters, meaning that if variables are sent as the parameters to a function, a copy of the variable is made for use in the function. This means that if a function changes the value of a parameter, the original variable is not affected. As seen previously with variables, the type of data stored in a parameter isn't specified, and JavaScript doesn't check what type of data is being sent, nor does it check the number of parameters that are sent in a function call. The previous example declared a function which expects 3 parameters, but the function can still be called with more or less parameters. If less parameters are sent than expected by the declaration, missing parameters are set to *undefined*.

Each function also contains an arguments object, which has an array of the parameters sent to the function. This allows the function to access all parameters used when the function is called, including when more parameters are sent than expected in the declaration. The previous average function could therefore be rewritten as below, without specifying the number of parameters. This function can then be used for any number of parameters.

```
<script type="text/javascript">
function average()
{
  var total = 0;
  for(index in arguments)
  {
    total += arguments[index];
  }
  return total / arguments.length;
}
document.write(average(5,6,7,8));
</script>
```

As well as length, arrays have several methods, or functions that can be called. The following table lists just some of the functions.

Function Name	Purpose
push()	Add an element to the end of an array.
pop()	Remove the last element from the array.
reverse()	Reverses the order of the array.
sort()	Sorts the array.

Table 8.4: Array Functions in JavaScript

8.6 getElementById()

So far this chapter has presented some basic programming fundamentals in JavaScript, but one of the key capabilities of JavaScript is the ability to interact with a webpage, even after it has loaded. Later chapters will discuss the Document Object Model, or DOM, where the page, or document, is considered as an object, comprised of other objects, indeed a tree of objects or elements. We have seen how the root element would be the html tag, and inside that tag are other elements including the head and the body tags. JavaScript can

change the way a page looks by adding, removing or editing elements within the DOM. Manipulating the DOM becomes an important function of JavaScript, and later chapters will investigate libraries that help with DOM manipulation. For now we will consider accessing specific elements on the page, and to assist with this, JavaScript comes with a very useful function; `getElementById()`.

`getElementById()` allows us to manipulate, or get data from any element on the page, by specifying an ID parameter. A variable can then be easily created referencing any element on the page;

```
var el = document.getElementById("test");
```

Once an element has been located, data about that element can be read into the code, or changed by the code. The data can include CSS information, parameters of the tag or text / HTML that appears within the tag. In the following example, a paragraph tag is created with the id "intro". Although this paragraph has no initial styling, when the JavaScript runs, it locates the element with the "intro" id, and changes the color to be blue.

```
<p id="intro">Hello World</p>
<script type="text/javascript">
  var el = document.getElementById("intro");
  el.style.color = "blue";
</script>
```

The `getElementById()` function is so useful that in many cases it is aliased, and replaced with the symbol "\$". As the function is used so frequently, it makes code simpler to use the alias, rather than the full function name. A simple function can be used to alias `getElementById()`, and similar functions are part of libraries such as jQuery.

```
<script type="text/javascript">
  function $(id)
  {
    return document.getElementById(id)
  }
</script>
```

Having aliased the `getElementById()` function, it can simply be referred to as \$, making a much simpler version of the previous example.

```
<p id="intro">Hello World</p>
<script type="text/javascript">
  $("intro").style.color = "blue";
</script>
```

In addition to changing CSS properties, the `innerHTML` property can be used to either return or change the content of an HTML tag, so for example the following code would change the text in the element with id intro to say Goodbye instead.

```
<p id="intro">Hello World</p>
<script type="text/javascript">
  $("intro").innerHTML = "Goodbye";
</script>
```

In each of these examples the code is run immediately when the page is loaded, so the user wouldn't notice that the text was originally black, or originally said "Hello World". In the coming example, we'll investigate running some code when the user interacts with the page by clicking on a button.

8.7 Form Validation Example

HTML's form tags can be used to create a form to get input from a user, and make the page more interactive. We will see later how the data inputted can be sent to the server, or another page. Generally a form will contain a submit button which can be clicked on by the user creating some kind of action. In this chapter we will look at how JavaScript can be used to validate that the form has been filled in correctly before it is submitted. Firstly we will create a simple form which just contains a single field for the user to input their name.

```
<form name="register" action="welcome.htm" onsubmit="return validate()" method="post">
  Username: <input type="text" name="username"><br />
  <input type="submit" value="Go!">
</form>
```

In this example, the form has 4 parameters. Firstly it has a name, which is needed so we can refer to it later. Secondly it has an action, which specifies a page to open once the form has been correctly filled in. Third there is the onsubmit parameter which makes a function call to a validate function, we shall shortly write this function in JavaScript. Finally the method specifies the way the action should be completed – we shall discuss this further later on. The form itself consists of an input text box and a submit button, so would look like this when opened in the browser.

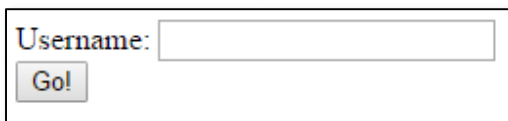


Figure 8.1: Simple Form for JavaScript Validation

Currently if you click on the "Go!" button, the browser will attempt to open a page called "welcome.htm", which hasn't yet been created, so will cause an error. Before we create a welcome page, first we want to validate that the form has been filled in correctly, or in other words, check that the username field isn't blank. To do this we can write the validate() function which is called when the user clicks on the submit button.


```
<script type="text/javascript">
function validate() {
    var name = document.forms["register"]["username"].value;
    if (name.length < 3) {
        alert("Username must have at least 3 characters");
        return false;
    }
}
</script>
```

Here the function creates a variable and assigns it the value from the username field. There is then a simple if statement to check if the username has at least 3 characters. If the username is less than 3 characters in length, an alert, or dialog box, is created and then the form is rejected by returning false. This example can be extended by adding more form elements, for example some radio buttons for the user to select their gender.

```
<form name="register" action="welcome.htm" onsubmit="return validate()" method="post">
    Username: <input type="text" name="username"><br />
    Gender:<br />
    <input type="radio" name="gender" value="male" /> Male
    <input type="radio" name="gender" value="female" /> Female<br />
    <input type="submit" value="Go!">
</form>
```

The input type for radio buttons is radio, and this will give the user the choice of male or female. The validate function can then be extended with a further if statement to make sure the user fills selects their gender, as follows.

```
var gender = document.forms["register"]["gender"].value;
if (gender=="")
{
    alert("You must select your gender");
    return false;
}
```

As this is a registration form, we may expect the user to select a password, fortunately the input tag has a type called "password", which will hide the users input.

```
Password:<input type="password" name="pass1" /><br />
Re-enter Password:<input type="password" name="pass2" /><br />
```

By adding this to the form, the user is asked to input their password twice. Further code can then be added to the validate function, to make sure that the password satisfies various conditions, for example checking that the two are the same, and have at least 8 characters.

```
var pass1 = document.forms["register"]["pass1"].value;
var pass2 = document.forms["register"]["pass2"].value;
if(pass1!=pass2)
{
    alert("Passwords must match!");
    return false;
}
if(pass1.length<8)
{
    alert("Passwords must have 8 characters");
    return false;
}
```

Try it yourself!

There are several ways that this example can be developed;

- * Make the form look good with CSS!
- * Indicate required fields with a "*", that disappears when it is filled out!
- * Ask for date of birth using a calendar!
- * Separating the JavaScript into a separate .js file!

Dive in and give it a go!

Key Points

- JavaScript is an important scripting language used to bring a page to life.
- JavaScript runs on the client machine, in the browser.
- JavaScript can be included in the header or body of a page, or in a separate .js file.
- The browser console is an essential tool for debugging JavaScript code.
- JavaScript is weakly (loosely) typed, so variables can store any kind of data.
- Control statements are used to manage the flow of a program, enabling branching and iteration.
- Arrays are used to store a collection of data.
- Functions are used to break your code down into manageable pieces.
- getElementById() is an important function which allows JavaScript to access and manipulate elements on in the document.
- HTML Forms can be used to transfer data from one page to another.

Further Resources

- 1) W3Schools has a JavaScript tutorial with plenty of examples and exercises to practice. There tutorial can be found here:-
<https://www.w3schools.com/js/>
- 2) You can try JavaScript examples out at javascript.com:-
<https://www.javascript.com/>
- 3) Codecademy has a free introductory JavaScript course, or a paid intensive course which can be found here:-
<https://www.codecademy.com/learn/learn-javascript>

Assignment

This chapter introduced a basic form validation script, it's your job to extend it! In Chapter 4 we created a simple form that could be validated by PHP. A better user experience is created if the input data is validated by JavaScript before sending it to the server. Create a form to register for a website. Your script should validate the following inputs in the same way as done in PHP;

- 1) Forename – Must not be blank, must not contain spaces, and must have at least 3 alphabet characters
- 2) Surname – Same rules as for Forename
- 3) Username – At least 5 characters and can include numbers, _ and –
- 4) Password – Must be at least 8 characters, containing both upper and lower case letters, numbers and symbols.
- 5) Age – The age must be between 18 and 110
- 6) Email – Must be of the form abc@def.ghi

Enhance your form to make sure the user re-inputs their password the same twice and hide the input from the view. Make your form look good with css – you can add a “*” next to required fields, that disappears when correct input has been put in. Figure out how to input the birthdate using a drop down calendar. Save your functions in a separate .js file.

Chapter 9

Introducing JQuery

Objectives

In this chapter we return to writing code to run in the browser, rather than on the server. JQuery is a powerful JavaScript library, which makes writing JavaScript much easier, allowing developers to create rich interactions for page visitors. JQuery allows the page to respond to the users actions on the page by handling events such as a mouse click, it also allows some interesting animations as elements on the page can be changed. After reading this chapter, you should be able:-

- To use JQuery functions on your webpages.
- To use selectors to find elements within the webpage document.
- To handle mouse, keyboard, form and browser events.
- To change a webpage with animation effects such as fade and slide.
- To dynamically change the css of any element on the page.
- To chain animations after each other, or use callback functions to call another function once an animation is completed.

Contents

9.1 Handling Events with JQuery

9.2 JQuery Effects

JQuery is a JavaScript library, designed to make writing JavaScript simpler and so it is useful for managing inputs and interactions with a page visitor, changing the way a page appears and interacting with the server. Whilst there are alternative JavaScript libraries, JQuery is the most popular. JQuery makes many aspects of using JavaScript simpler, including traversing and manipulating the HTML document (DOM), handling events and animations, and managing AJAX, by creating a simpler API (Application Programming Interface) for developers to use.

JQuery is a JavaScript library, designed to make writing JavaScript simpler, useful for traversing and manipulating the HTML document (DOM), handling events and animations and managing AJAX by creating a simpler API for developers to use.

Because JQuery is simply a JavaScript library, it is contained within a .js file. This file can be downloaded from jquery.com, and then included in the head of any html document;

```
<head>
  <script src="js/jquery-3.1.0.min.js"></script>
</head>
```

This allows you to host the jQuery library on your server, or on your local machine, which is particularly useful for developing perhaps when you are disconnected from the web. An alternative though is to use one of the versions already hosted on a CDN (Content Distribution Network), perhaps the version hosted by google. In this case just add a link to Google's version of the script.

```
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>
</head>
```

There are several advantages to using the version of jQuery hosted by google. Firstly the download time is likely to be reduced – as the file is hosted on a CDN, the nearest copy will be delivered to the user, i.e. if they are in Europe they will get a European copy, while if they are in Asia they will get an Asian copy, without needing to directly contact your server. This will also reduce the workload for your server, and as a free service – why not? Also there is a reasonable chance that the user will already have visited a site that also uses google's version of JQuery, in which case there may be a copy cached in the user's browser. A further benefit is *increased latency*, as the browser can download files from different servers concurrently. Of course if the CDN happens to be down, or you are missing an internet connection, your site may not work as you expect it to.

The basic syntax for a JQuery expression involves the "\$" function, which is similar to the "getElementById()" function introduced in the JavaScript chapter. This function allows you to select HTML elements from a document and perform some action upon them. The basic syntax is as follows;

`$(selector).action()`

In this case the \$ represents JQuery and the getElementById() function. “selector” represents any particular document element, class or id, and action represents any of the predefined jQuery functions that can be performed on a selector. An example of an action is the “hide()” function, which makes an element disappear, in reality this is the equivalent of setting the css property of an element to “display:none”, hence making it disappear. The following give examples of the hide() function, where first the hide function is called for the paragraph selector (p), which would make any elements that are in paragraphs disappear. Secondly the element with the class “test” would disappear – notice the “.” indicating that a class is being referenced. Thirdly the element with the id “test2” would disappear – notice the “#” used to reference an id. Finally “this” could be used to reference a selector that had already been selected.

```
$(“p”).hide();
$(“.test”).hide();
$(“#test2”).hide();
$(this).hide();
```

This illustrates how jQuery allows the developer to find and select specific HTML, based on their id, their class, or attributes, or values of attributes, etc. This makes the \$ function, more powerful than simply getElementById(), as it returns a jQuery object, upon which a variety of jQuery functions can be called. JQuery functions may be saved in a separate .js file which can be loaded in the header, therefore it is often important that the page elements are loaded before a function is defined. This is generally done by placing jQuery code within a function that is called after the document is loaded, or rather after the document is ready.

```
$(document).ready(function() {
    console.log(“Loaded”)
});
```

The document.ready() function makes sure that the page is loaded before jQuery functions are called upon it, meaning (amongst other things) elements that haven’t been loaded won’t be hidden until they are. Notice that a function is defined within the document.ready() function, and notice the sequence of different bracket delimiters needed to complete the code. In this case, the code would log “Loaded” into the console once the page was ready.

There are a wide variety of ways that selectors can be used to specify particular page elements and the following code illustrates some options for selecting different page elements.

```

$(document).ready(function() {
  $("*") //Selects ALL elements on the page
  $("p.intro") //Selects paragraph elements with the 'intro' class
  $("p:first") //Selects the first paragraph element
  $("ul li:first") //Selects the first list item (li) in an unordered list (ul)
  $("[href]") //Selects elements that have a href attribute
  $("a[target='_blank']") //Selects links where the target attribute is _blank
  $("a[target!='_blank']") //Selects links where the target attribute is NOT _blank
  $("tr:even") //Selects even rows in a table
  $("tr:odd") //Selects odd rows in a table
});

```

9.1 Handling Events with JQuery

There are a variety of events that jQuery can respond to, such as when a page visitor creates some action on the page, like moving a mouse over an element or clicking on an element. Previously we saw how a JavaScript function could be called when the user clicked on a submit button, but in this case the JavaScript function call was embedded within the form. Using jQuery all the programming logic for handling events, such as submitting a form, can be separated from the document elements. Separating the behavior of a page from the contents of a page makes the coding easier both for developing the code, and for maintaining the code. The main events that can occur on a webpage can be separated into mouse events, keyboard events, form events and document events.

Mouse Events	
click	When the user clicks on the element
dblclick	When the user double clicks on the element
mouseenter	When the user moves the mouse over the element
mouseleave	When the user moves the mouse away from the element
Keyboard Events	
keydown	When the user presses down a key on the keyboard
keyup	When the user releases a key on the keyboard
keypress	When the user types a key
Form Events	
submit	When a form is submitted
change	When the value of an input is changed
focus	When a form input element becomes active
blur	When a form element stops being active
Browser Events	
load	When the page finishes loading
resize	When the page is resized
scroll	When the page is scrolled
unload	When the page is closed

Table 9.1: Events that can be Handled by JQuery

9.1.1 Mouse Events

The mouse events revolve around where the mouse is moved to on the page, with the mouseenter and mouseleave events occurring when the mouse is moved over a particular element. The click and dblclick events occur logically when the user clicks or double clicks on an element. Let's look at a complete example. Here the jQuery library is included in the head, followed by some jQuery code. Notice that the body contains a heading, 2 paragraphs and a button, but contains no script code at all, not even a function call. From here all code can be separated from the content of the page. The jQuery script first waits for the document to be ready, and then adds some code to the click event for the "button" element. When the click event occurs, a function is called, which hides any paragraph elements.

```
<html><head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
<script>
  $(document).ready(function(){
    $("button").click(function(){
      $("p").hide();
    });
  });
</script>
</head><body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>Click me</button>
<br>
</body></html>
```

In this second example, we just look at the jQuery code within the header. In this example the "dblclick" event is used, and this time when the button element is double clicked the element which has the id "test" is hidden. Note that this function is added to the page after the document is ready, and a further function is defined within the dblclick event.

```
<script>
$(document).ready(function(){
  $("button").dblclick(function(){
    $("#test").hide();
  });
});
</script>
```

With a further example the syntax should become more familiar. This time the event is the "mouseenter" event, so is initiated when the mouse hovers over the element that has the "myelement" id.

This time the \$ function is used to identify any paragraph elements that have the test class, and then the hide() function is called to make them invisible.

```
<script>
$(document).ready(function(){
  $("#myelement").mouseenter(function(){
    $("p.test").hide();
  });
});
</script>
```

Essentially the jQuery mouse events can be called for whenever the mouse interacts with any element on the page, that can be identified by any of the selectors introduced previously. In this example, an alert box is created when the user moves the mouse away from whatever element has the id "p1".

```
<script>
$(document).ready(function(){
  $("#p1").mouseleave(function(){
    alert("Bye! You have now left p1!");
  });
});
</script>
```

9.1.2 Keyboard Events

There are 3 important keyboard events, keydown, keyup and keypress. As with all of events, the functions written to deal with the events are event handlers. When typing the action of pressing a key can be separated between the event of the down press and the event of releasing the key, and so different actions can be coded for each event. This example concerns when a user types in an element with the id "input". When a key is pressed, the element with id "display" is hidden, and when the key is released the element reappears.

```
<script>
$(document).ready(function() {
  $("#input").keydown(function() {
    $("#display").hide()
  });
  $("#input").keyup(function() {
    $("#display").show()
  });
});
</script>
```

The 3rd keyboard event is the keypress event, which is very similar to the keydown event, except that pressing non-printing keys such as shift, delete or escape would trigger the keydown event, but as they don't

cause the browser to register a keyboard input, they wouldn't cause the keypress event. The keypress event also isn't officially specified, which means different browsers may respond differently to keypress events. Clearly it is useful to identify which key has been pressed, and this can be done using the event object passed as a parameter to the function. The event object has a "which" property that can be used to identify the character code of the key that was pressed.

```
<script>
$(document).ready(function() {
  $("#input").keypress(function(event) {
    alert(event.which);
  });
});
</script>
```

Standard programming logic can be used within the jQuery function, as in the following example which stops the default action when the user presses the enter key, which has the character code 13.

```
<script>
$(document).ready(function() {
  $("#input").keypress(function(event) {
    if(event.which==13)
    {
      alert("You pressed enter!");
      event.preventDefault();
    }
  });
});
</script>
```

9.1.3 Form Events

Form events are used to handle interactions with the form, such as when a user submits a form. In chapter 4 we added a function call to the onsubmit parameter for a form to use JavaScript to validate a form;

```
onsubmit="return validate()"
```

Using jQuery we can move the programming logic away from the HTML form, by adding code to the submit event. Separating the programming logic away from the form creates more versatile code, and makes life easier for designers working alongside coders. In this example, when the form with the id "input" is submitted, an alert is produced as well as a call to a validate function, before the default event of submitting the form to the server is prevented.

```
<script>
$(document).ready(function() {
  $("#input").submit(function(event) {
    alert("Form Being Validated!");
    validate();
    event.preventDefault();
  });
});
</script>
```

The other form events, “focus”, “blur” and “change” can be used to create code to handle when an element becomes the active field in a form (focus), or stops being the active field on a form (blur), or when the value of an element is changed (change).

9.1.4 Browser Events

As we’ve seen, JQuery allows code to be added to a variety of events allowing the page to respond to user interaction. As well as for managing user interactions, jQuery also has events that may not be triggered by the user, such as the document ready function which as we have seen responds to the event when the document is fully loaded. Similar page events can be handled such as when the page loads, unloads, resizes or is scrolled, using the browser events.

9.2 JQuery Effects

The examples thus far have largely simply responded to events by making other selectors disappear, but jQuery has many, much cooler effects that can make the page come to life. Of course the basic hide() function works by setting the css property of an element to “display:none”, while the show() function is roughly similar to setting the css property of an element to “display:block”, although technically it will revert the display property to its previous setting. The hide and show functions can be made more interesting though by adding parameters. The basic syntax of the hide function is;

```
$(selector).hide(speed, callback);
```

The speed parameter could be “slow”, “fast” or an integer reflecting the number of milliseconds it should take to disappear. If an element is set to hide (or show) using the parameter ‘slow’, it will take 200 milliseconds, while using the parameter ‘fast’ will take 600 milliseconds. While previously the element would disappear instantly, by setting the hide function to run over time it allows for animations. The callback is a further function to call after the animation has completed – more about this later in the chapter. As the hide and show functions switch the state of an element between visible and not visible, it is possible to lose track of whether an element is visible or not. A further function “toggle()” will change the current state of the element depending on whether it is visible or not.

The following example demonstrates how a paragraph element can be hidden over the period of a second (1000 milliseconds), after a button is clicked.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").toggle(1000);
  });
});
</script>
```

9.2.1 Fade Effects

Within CSS the opacity of an element can be set to specify the transparency of an element. The default value for opacity is 1, where elements are completely visible or solid, however the opacity can be set between 0 and 1, where 0 is completely see through. This example demonstrates 4 different levels of opacity for a plain black box.

```
<html><head>
<style>
span {
  height: 100px;
  width: 100px;
  background: #000;
  float:left;
}
span.opacity100 { opacity: 1; }
span.opacity70 { opacity: 0.7; }
span.opacity40 { opacity: 0.4; }
span.opacity10 { opacity: 0.1; }
</style>
</head><body>
<span class="opacity100">opacity: 1</span>
<span class="opacity70">opacity: 0.7</span>
<span class="opacity40">opacity: 0.4</span>
<span class="opacity10">opacity: 0.1</span>
</body>
</html>
```

The result of this displays 4 boxes with different opacities. On the surface it appears similar to a previous example, as though there are 4 boxes with different colours, but in this case anything behind the elements could be seen through.

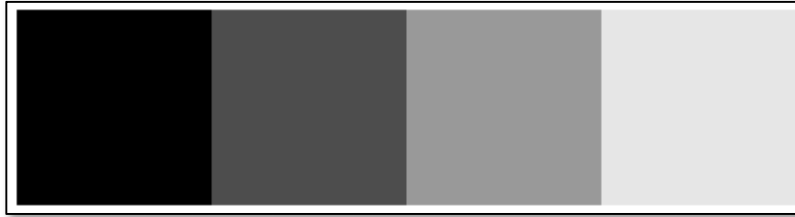


Figure 9.1: Opacity of Page Elements

The fade effects make use of the opacity property of elements, by gradually changing the opacity over time to make it appear as though an element is fading in or fading out. There are 4 different fade related effects.

Fade Effects	
fadeIn()	Gradually changes opacity until the element is completely transparent
fadeOut()	Gradually changes opacity until the element is completely opaque
fadeToggle()	Switches between opaque and transparent
fadeTo()	Changes opacity to a specified level

Table 9.2: Fade Effects in JQuery

The fadeIn(), fadeOut() and fadeToggle() functions all take a parameter specifying either “slow”, “fast” or a time in milliseconds for the transition, and can also specify a callback function. The following example will fade the 2 specified divs at different speeds.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").fadeOut("fast");
    $("#div2").fadeIn("slow");
    $("#div3").fadeToggle(3000);
  });
});
</script>
```

The fadeTo() function is even more useful allowing the developer to specify the level of opacity for the element to fade to – either increasing the opacity of decreasing it. This function takes 2 parameters, both the speed of the fade, and the resulting opacity level.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").fadeTo("slow",0.15);
    $("#div2").fadeTo("slow",0.4);
  });
});
```

```
});  
</script>
```

9.2.2 Slide Effects

While the fade effects gradually change the opacity property of an element, slide effects gradually change the height property of an element, which gives the effect of an element sliding in or out. If an element slides in, by changing its height property, later elements in the page will move down to make space.

Slide Effects	
slideDown()	Slides an element down, by increasing its height
slideUp()	Slides an element up, by decreasing its height
slideToggle()	Switches the state of an element between up and down

Table 9.3: Slide Effects in JQuery

The next example, with limited css, shows a menu div which when clicked upon displays a slide down panel.

```
<html><head>  
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>  
<script>  
  $(document).ready(function(){  
    $("#menu").click(function(){  
      $("#panel").slideDown("slow");  
    });  
  });  
</script>  
<style>  
  #panel,#menu { border:solid 1px #333; }  
  #panel { display:none; }  
</style>  
</head><body>  
<div id="flip">Click Here</div>  
<div id="panel">Hello world!</div>  
</body></html>
```

The same example can be extended using the slideToggle() function to slide the panel up and down.

```
<script>  
  $(document).ready(function(){  
    $("#menu").click(function(){  
      $("#panel").slideToggle("slow");  
    });  
  });  
</script>
```

9.2.2 Animation Effects

Animations can be created by modifying css properties of elements over a period of time, perhaps by gradually changing the position, size or colour of page elements. Later we shall investigate jQuery's "css()" function, but first jQuery also has an "animate()" function, which provides an easy interface for changing css properties over a period of time.

```
$(selector).animate({params}, speed, callback);
```

This function takes several parameters including first the params, or css properties that should be changed, secondly the speed of the change, and finally a callback function to perform after the animation completes. In this example, when the button is clicked, any div element is animated by moving it so that the left property is 250 pixels, or 250 pixels from the left of its containing element.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({left:'250px'});
  });
});
</script>
```

More than one css property can be changed at the same time as in the following example, where the div element is resized (to a 150 pixel square), moved and made 50% opaque.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({
      left:'250px',
      opacity:'0.5',
      height:'150px',
      width:'150px',
    });
  });
});
</script>
```

The css property changes could also be relative values, whereas in the previous examples the size and position of the element after the animation is set absolutely to 150 pixels size, and 250 pixels from the left, this could be relative value when compared to its current position and size. In the next example the element is

moved in the same way, but the size of the element is adjusted to be 150pixels bigger, which means each time the button is clicked the element will get bigger and bigger.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({
      left:'250px',
      height:'+=150px',
      width:'+=150px'
    });
  });
});
</script>
```

As well as coding animations to happen at the same time, animations can also happen sequentially. The next example differs slightly, as first a variable is created to store the div, then 4 calls to the animate function are called, one after another.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    var div=$("#div");
    div.animate({height:'400px',opacity:'0.2'},"slow");
    div.animate({width:'400px',opacity:'1'},"slow");
    div.animate({height:'50px',opacity:'0.2'},"slow");
    div.animate({width:'50px',opacity:'1'},"slow");
  });
});
</script>
```

This example works as expected because each of the functions is called on the same element, however when dealing with timing events the sequence of events can become confused, such as in the following example.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").hide("slow");
    alert("All Gone!");
  });
});
</script>
```


In this case the alert will be shown at the same time as the hide animation begins, which would create the alert prompt before the div has been hidden. It is this kind of scenario where a **callback function** is useful. A callback function is called after the current effect is complete, for example after the current animation completes. In the following example, the callback function is called to create the alert after the div is hidden.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").hide("slow",function(){
      alert("All Gone!");
    });
  });
});
</script>
```

Another related feature of jQuery is the ability to chain functions together one after another. In this example each function call may as well be on the same line with each being called sequentially on the same div element.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({height:'400px',opacity:'0.2'},"slow")
    .animate({width:'400px',opacity:'1'},"slow")
    .animate({height:'50px',opacity:'0.2'},"slow")
    .animate({width:'50px',opacity:'1'},"slow");
  });
});
</script>
```

Key Points

- JQuery is a powerful JavaScript library that makes writing client side code easy.
- Selectors are used to identify elements within a webpage.
- JQuery code is written to respond to events when a user interacts with a webpage, such as when they click on a particular element.
- JQuery can be used to respond to events by changing the document, through animation effects such as fading, sliding or changing other CSS properties.

Further Resources

- 1) JQuery can be downloaded from the main JQuery website, which also has excellent reference materials, here:-
<https://jquery.com/>
- 2) w3schools has a tutorial / guide that introduces JQuery, which can be found here:-
<https://www.w3schools.com/jquery/>

Assignment

JQuery can make your calendar from the previous chapters more attractive – write code so that the box for each day changes colour each time the mouse hovers over it and be imaginative about how else the calendar can be more user friendly.

Revisit the form validation assignment from chapter 8, but use JQuery to give the user more help when filling in the form by highlighting required fields that haven't been filled in correctly.

Chapter 10

JQuery and the DOM

Objectives

This chapter introduces the Document Object Model, or DOM. The DOM presents a different way of visualizing a webpage, or document, where each element in the document is an object that is interlinked with the other elements in a tree-like structure. JQuery offers great tools to manipulate the DOM, by adding or removing elements from the DOM, or traversing the DOM to find other selectors. After reading this chapter you should:-

- Understand how a webpage can be seen as a document composed of interrelated objects.
- Be able to change text, html, values and attributes on a page.
- Be able to add and remove elements in different ways from the DOM.
- Be able to traverse the DOM, moving up, down or across to locate other elements.
- Be able to use filters to change the selected DOM elements.

Contents

10.1 The Document Object Model

10.2 Manipulating the Document Object Model

10.3 Navigating the Document Object Model

The DOM, or Document Object Model, is a structured model of a webpage, illustrating how various parts of the page fit together. A webpage is a document, which is comprised of a number of elements, or objects, which are all interconnected. These objects each have properties and capabilities. By defining the webpage as a model of objects, it creates a programming interface that can be used to manipulate it, in this case JavaScript, or jQuery, is used to navigate and edit elements in the DOM.

10.1 The Document Object Model

A webpage is a document, written using HTML which instructs the browser the content and structure of the page, as well as how it should be styled. While the browser parses a version of the document, the DOM provides a neutral interface for scripts to dynamically access and change the content, structure and style of the document.

The DOM is a structured model of a webpage. While the browser parses a version of the document, the DOM provides a neutral interface for scripts to dynamically access and change the content, structure and style of the document.

The DOM can be visualized as a tree of interrelated nodes, with the document being the root node, and elements on the page being children nodes, beginning with the HTML tags. Consider the following very simple HTML page.

```
<html>
<head>
  <title>Simple Page</title>
</head>
<body>
  <div class="simple">
    <h1>My Heading</h1>
    <p>My Paragraph</p>
  </div>
</body>
</html>
```

The root of the DOM is the document, which contains the html tag as a “child” beneath it in the tree. Within the HTML element, there are 2 subparts, the head and the body. The resulting document could be visualized then in several ways. Firstly, the view rendered by the browser after parsing the html document, which would appear similar to this.



Figure 10.1: Sample Page to Illustrate the DOM

An alternative way of visualizing it, would be as a tree like structure. Here the document is considered the root node, with the html tag being its child. The html tag has 2 further children; the head and body tags. These tags have further children until leaf nodes are reached, such as the paragraph leaf node which only contains text. Clearly this is a simple webpage with few elements – a more complex webpage could still be visualized as a tree which contains more nodes / elements. When a tag is nested inside another tag, it becomes a child of that tag.

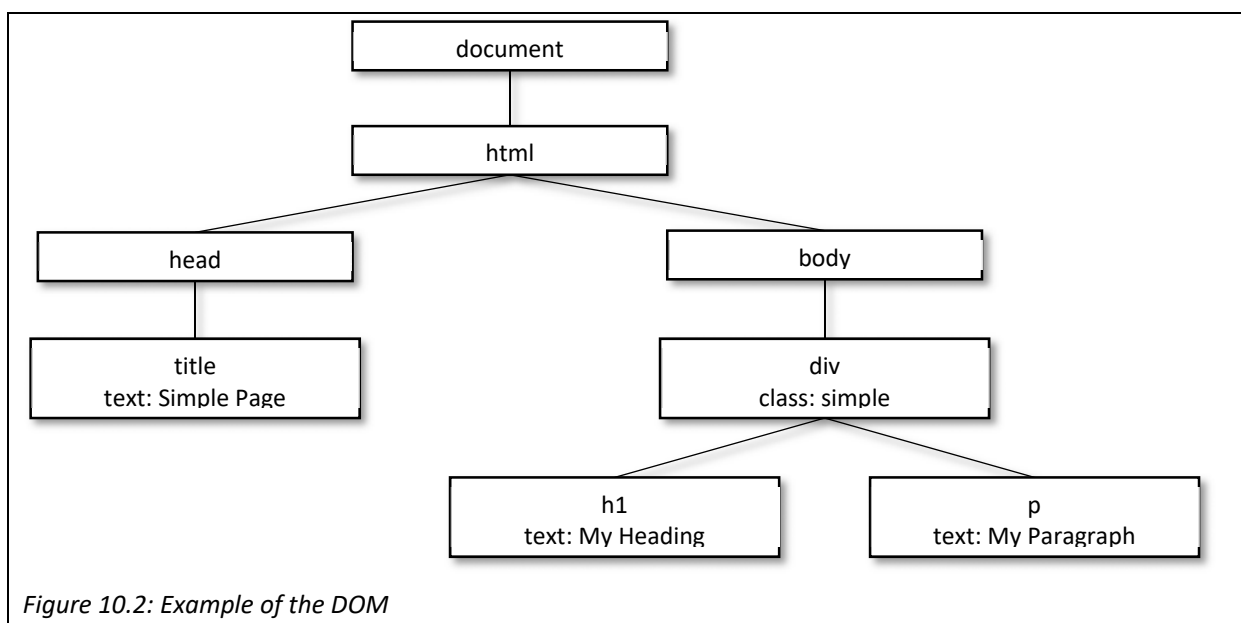


Figure 10.2: Example of the DOM

A further example of this would be an unordered list, using the `` tag. Within the opening and closing `` tags, a number of list items can be created, each inside `` tags. Each of the list items naturally become children of their parent `` tag.

Another way of visualizing the DOM is as a textual version of the tree.

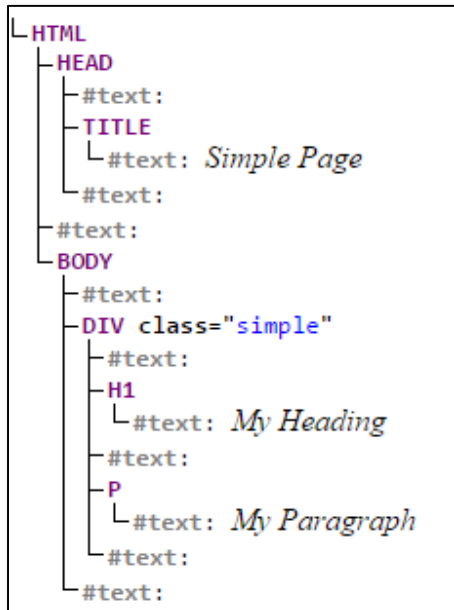


Figure 10.3: Textual Version of the DOM

10.2 Manipulating the Document Object Model

jQuery offers some useful functions for navigating through the DOM, and editing the contents. We could use jQuery to identify the paragraph element in the previous example, and then we could change the text displayed within the paragraph. There are several jQuery functions that are useful for accessing, and changing parts of the DOM, and given that this can be controlled by script in the browser, it means the page can be changed after it has loaded, and based on user interactions with the page. In this section we look at the `text()`, `html()`, `val()` and `attr()` functions.

10.2.1 Manipulate Text

The “`text()`” function will either return the current text contents of an element, or change them, depending on whether a parameter is included or not. In this example an alert will display the contents of the element that has the id “`myparagraph`”, after the button is clicked.

```

<script>
$(document).ready(function(){
  $("#button").click(function(){
    alert($("#myparagraph").text());
  });
});
</script>

```

A quick edit of this script can allow the script to change the content of the “`myparagraph`” element by including a parameter containing the new text.

```

<script>
$(document).ready(function(){
  $("#button").click(function(){
    $("#myparagraph").text("Changed the text to this");
  });
});
</script>

```

10.2.2 Manipulate HTML code

HTML of course is a combination of text and tags, and while the `text()` function will return the text contained in an element, the `html()` function will return any html within the element, including other html tags. In this case the `text()` function would return "Some text", while the `html()` function would return "`Some text`".

```

<script>
$(document).ready(function(){
  $("#button").click(function(){
    alert($("#myparagraph").text());
    alert($("#myparagraph").html());
  });
});
</script>
<p id="myparagraph"><strong>Some text</strong></p>

```

Of course the `html()` function can also change the html within an element, simply by adding a parameter to the function call. In this case the html within the paragraph tag would be changed to display "Hello World" within strong tags.

```

<script>
$(document).ready(function(){
  $("#myparagraph").html("<strong>Hello World</strong>");
});
</script>
<p id="myparagraph">Hello World</p>

```

10.2.3 Manipulate Values

The `val()` function is used to manipulate the values stored in an input field, such as a text box. Again, the function will either return or set the value. This can be particularly useful for validating a form, by accessing the value and checking that it meets some specified requirement. In this example the field with the id "pass1" is checked to make sure that it has at least 8 characters.

```

<script>
$(document).ready(function(){
  $("#submit").click(function(){
    if($("#pass1").val().length < 8) { alert("Password must be at least 8 letters long"); }
  });
});
</script>

```

10.2.4 Manipulate Attributes

As we have seen, elements may contain text, and they may contain further html. Some elements may also have a value. HTML elements also have attributes, which are assigned within the tags. These attributes can be manipulated using the attr() function. In this example the link with the id "mylink" is located and the href and title attributes are changed.

```

<script>
$(document).ready(function(){
  $("#submit").click(function(){
    $("#mylink").attr({
      "href" : "http://www.google.com",
      "title" : "Google"
    });
  });
});
</script>

```

10.2.5 Adding Elements to a Page

As well as allowing the developer to change existing HTML elements, jQuery also affords the ability to add and remove elements from the DOM. There are 4 functions in jQuery used to add elements to a page, all of which add an element in relation to an existing element that has been located using the \$ function. They are before(), after(), prepend() and append().

Adding to a Page	
before()	Adds content before the specified element
after()	Adds content after the specified element
prepend()	Adds content at the start of the specified element
append()	Adds content at the end of the specified element

Table 10.1: Adding Elements to a Page with JQuery

In the following example, the page begins with just a simple div with id "mydiv", which contains the text "Original Text". However when the document is ready, 4 function calls are made, one to each of the functions described above, to demonstrate exactly where the content would be added.


```

<html><head><title>Add</title>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('#mydiv').before("BEFORE");
    $('#mydiv').after("AFTER");
    $('#mydiv').prepend("PREPEND");
    $('#mydiv').append("APPEND");
});
</script>
</head><body>
<div id="mydiv">Original Text</div>
</body></html>

```

When loaded the page looks like this;

```

BEFORE
PREPENDOriginal TextAPPEND
AFTER

```

Figure 10.4: Demonstrating before(), prepend(), append() and after() in JQuery

This is perhaps better explained by inspecting the body element after the code has run.

```
BEFORE<div id="mydiv">PREPENDOriginal TextAPPEND</div>AFTER
```

10.2.6 Removing Elements from a Page

There are 2 further functions which can be used to remove elements from a page, these are remove() and empty(). The difference between the two functions is that the remove() function will completely remove an element, and delete all of its children elements, whereas with the empty() function, the contents of the element are removed, but the element remains.

10.2.7 Changing the CSS of Elements

There are also several ways to change the CSS of an element. We have already seen how the attr() function can be used to return, or set, the attributes of a page element, and this of course could be used to set the class attribute of an element. However there are alternatives, such as the addclass(), removeclass() and css() functions. The addclass() function just adds a class to any specified element.

```

<script>
$(document).ready(function(){
  $("#mydiv").addClass("subheading");
});
</script>

```

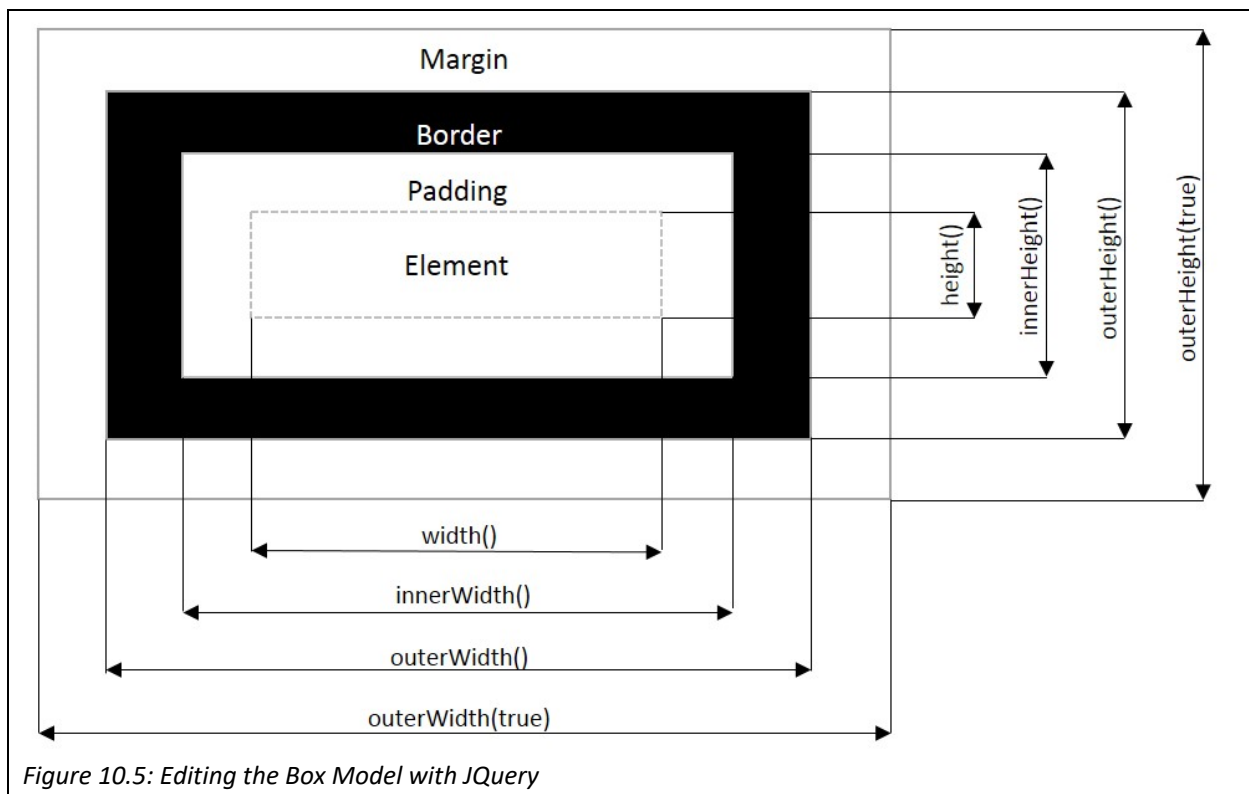
So while the `addClass()` and `removeClass()` have the ability to add styles from a separate stylesheet, the `css()` function allows styles to be added inline, specific to a particular element. In this example, the background color for any paragraphs will be set to green, when the button is clicked.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").css("background-color","green");
  });
});
</script>
```

In this case the style is added inline, with the resulting element looking like this when inspected.

```
<p style="background-color: green;">This is a paragraph.</p>
```

CSS is of course used for layout as well as style. In the previous chapter we investigated how jQuery can be used to move elements around the screen dynamically. jQuery also provides means to manipulate the dimensions of elements. The `width()` and `height()` functions allow access to get, or set the width and height of an element. Each element is naturally surrounded by padding, a border and a margin, and these can be controlled by CSS properties. jQuery also provides functions to return the true sizes of an element after padding, borders and margins are taken into consideration.



Whilst the `width()` and `height()` functions return the exact dimensions of the element, an alternative is the `innerWidth()` and `innerHeight()` functions, which return the dimensions of the element including its padding. The `outerWidth()` and `outerHeight()` functions return the dimensions including the padding and border, while adding a parameter to the `outerWidth(true)` and `outerHeight(true)` functions also includes the margin around the element.

10.3 Navigating the Document Object Model

As stated previously, the DOM is a treelike representation of the contents of a webpage. Using scripting languages like jQuery, the page becomes dynamic, changing over time, and so also the DOM is dynamic with nodes (elements) being added and removed. One of the most powerful features of jQuery is the ability to navigate and traverse the DOM, moving from one page element and locating other page elements, even after the page has transitioned through elements being added and removed. The DOM traversal functions in jQuery include functions for moving up the tree, functions for moving down the tree, and functions for traversing across the tree.

10.3.1 Traversing UP the DOM

The key function here is the `parent()` function, which given one page element, allows jQuery to identify the parent element, or containing element. For the following examples, the following page is used.

```
<html><head>
<style> .box { display: block; border: 2px solid #ddd; padding: 2px; margin: 2px; } </style>
</head><body>
  <div class="box grandparent" id="grandparent">Grandparent
    <div class="box" id="parent">Parent
      <div class="box" id="child1">Child 1</div>
      <div class="box" id="child2">Child 2</div>
      <div class="box" id="child3">Child 3</div>
      <div class="box" id="child4">Child 4</div>
    </div>
  </div>
</body></html>
```

When displayed the page looks as follows, with a grandparent div that contains a parent div, which contains 4 further child divs, each of which is styled with a grey border as shown in figure 10.6.

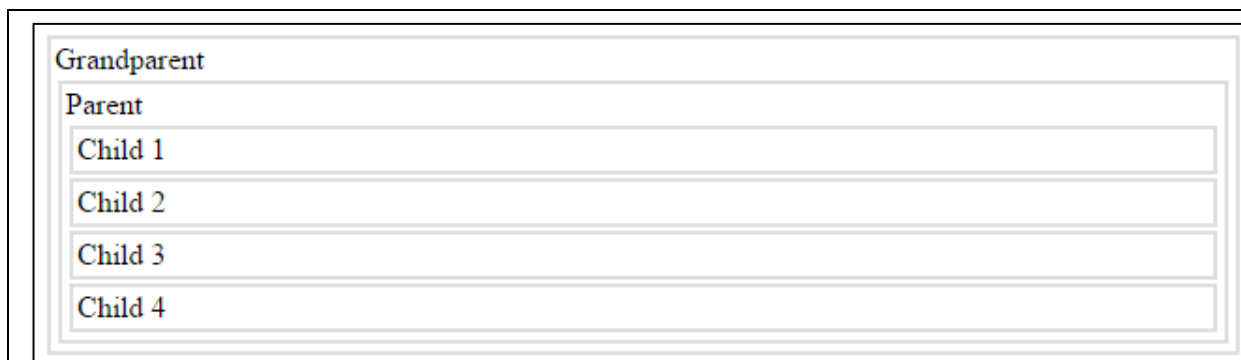


Figure 10.6: Sample Page for Demonstrating DOM Navigation

Now if we add the following script which first identifies the element in the document that has the id “child1”, and then navigates to the parent of this element. Once the parent has been located, the css of the border is highlighted.

```
<script>
$(document).ready(function() {
  $('#child1').parent().css({"border":"2px solid black"});
});
</script>
```

The result is as follows.

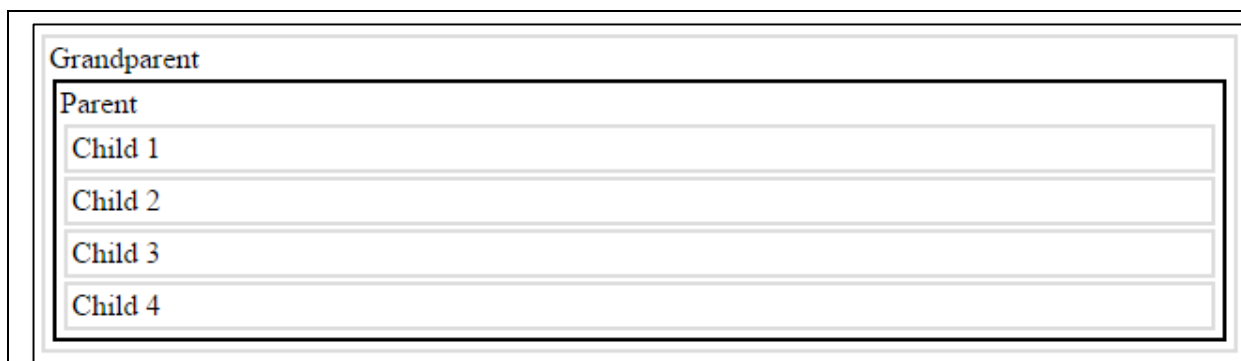


Figure 10.7: The parent() Function in JQuery

An alternative would be to change this to use the parents() function, which selects all parents of the element, including grandparents etc.

```
<script>
$(document).ready(function() {
  $('#child1').parents().css({"border":"2px solid black"});
});
</script>
```

Notice that now, not only does the parent have a black border, but also the grandparent. There are a further 2 borders outside of the grandparent, one is for the body element, and the outer one for the document, or html element.

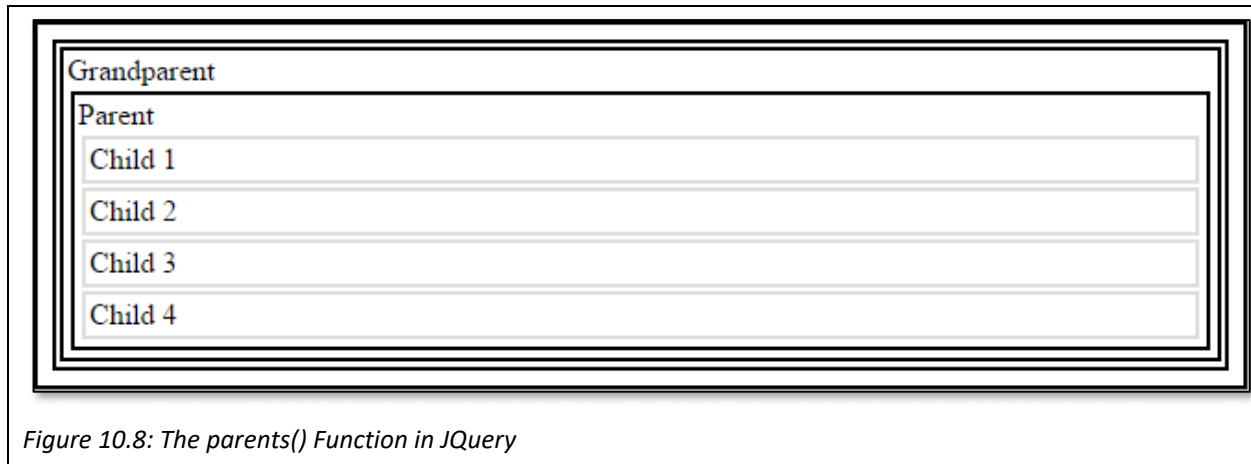


Figure 10.8: The `parents()` Function in jQuery

This can be further refined by using the `parentsUntil` function, where parents can be identified, until parents with a certain selector, such as the body.

```
<script>
$(document).ready(function() {
  $('#child1').parentsUntil("body").css({"border":"2px solid black"});
});
</script>
```

The result is as follows.

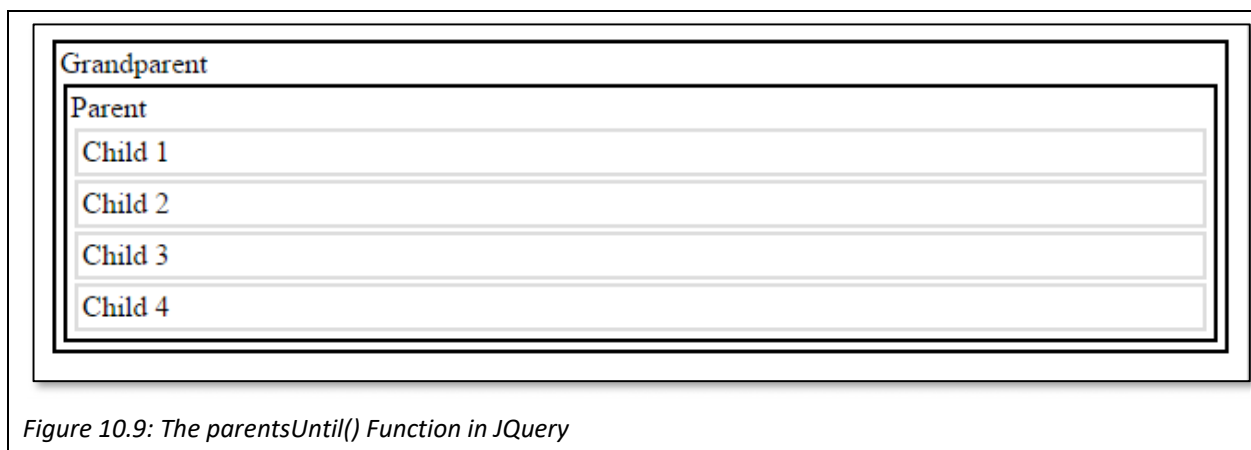


Figure 10.9: The `parentsUntil()` Function in jQuery

An alternative which results in exactly the same result, would be to add a parameter to the `parents()` function, specifying to only select parents that are "div" elements.

```

<script>
$(document).ready(function() {
  $('#child1').parents("div").css({"border":"2px solid black"});
});
</script>

```

10.3.2 Traversing DOWN the DOM

Just as jQuery enables traversing up the tree to parents, it also affords the ability to traverse down the tree, using 2 key functions; `children()` and `find()`. The `children` function will highlight all the immediate children for the selected element, so if it was called for the grandparent, then the parent element would be identified.

```

<script>
$(document).ready(function() {
  $('#grandparent').children().css({"border":"2px solid black"});
});
</script>

```

In this case, only the parent element is identified.

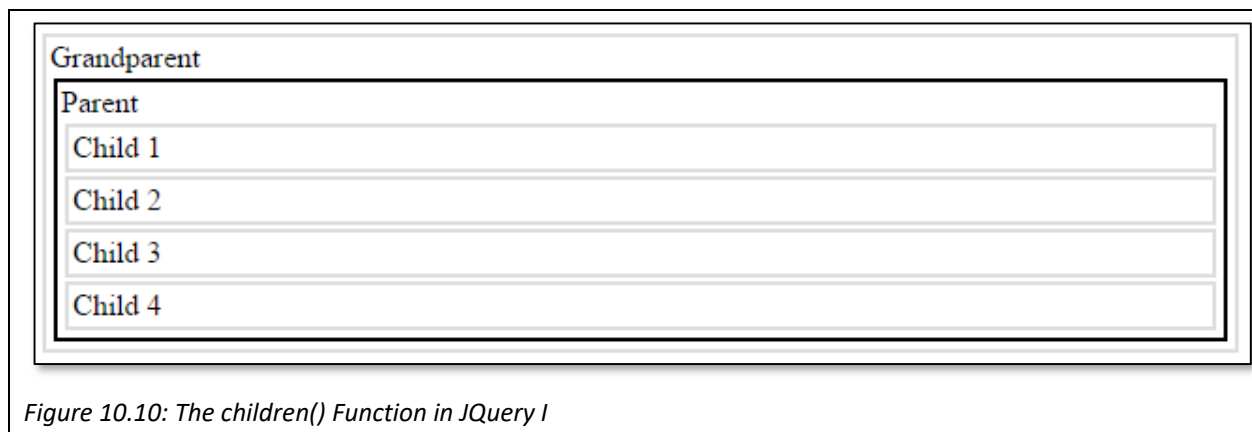


Figure 10.10: The `children()` Function in JQuery I

However, if the same function was called for the parent, or for the children of the children, then all the child elements would be selected.

```

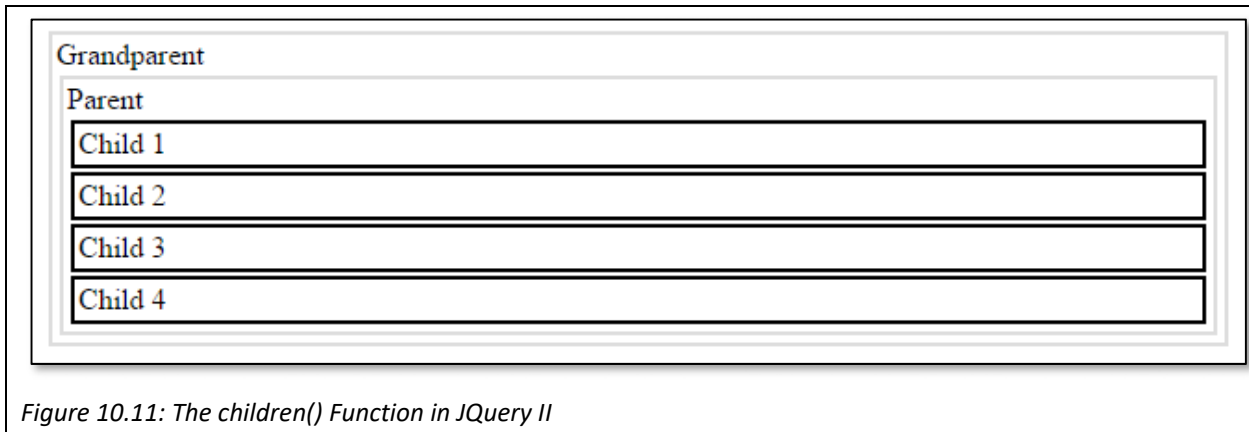
<script>
$(document).ready(function() {
  $('#grandparent').children().children().css({"border":"2px solid black"});
});
</script>

```

This has the same effect as;

```
<script>
$(document).ready(function() {
  $('#parent').children().css({"border":"2px solid black"});
});
</script>
```

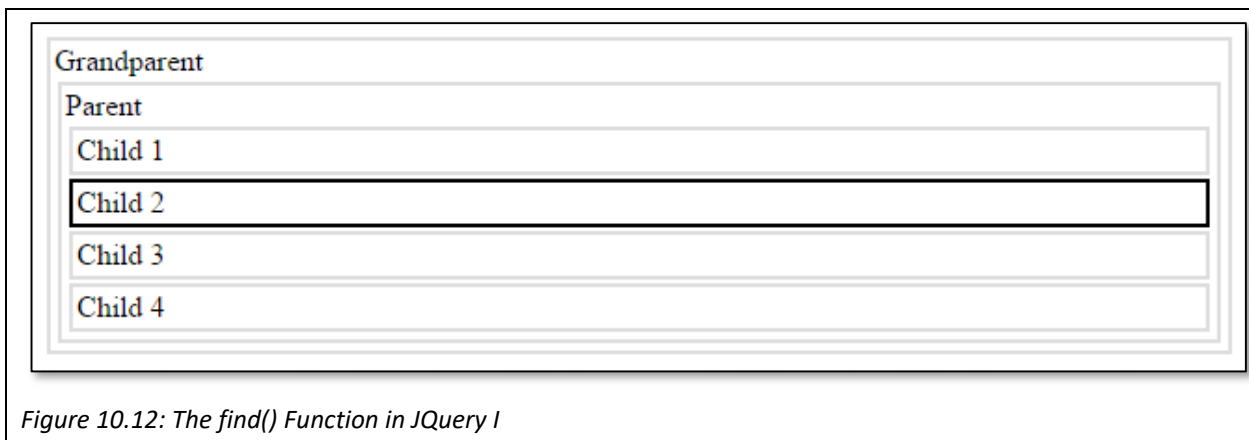
Both examples will select all 4 children elements.



The second function for traversing down the DOM is the find() function, which searches through an elements children, grandchildren, etc. for an element that matches a specified condition, for example, we could search from the grandparent for any lower level elements that have the id "child2".

```
<script>
$(document).ready(function() {
  $('#grandparent').find("#child2").css({"border":"2px solid black"});
});
</script>
```

The result from this script would be.



The selector in the parameter sent to the find() function could be any valid selector, so naturally an html or class selector would be a more logical selector than an id selector. Alternatively the wild card selector could be used to select all lower level elements.

```
<script>
$(document).ready(function() {
  $('#grandparent').find("*").css({"border":"2px solid black"});
});
</script>
```

The resulting page is as follows.

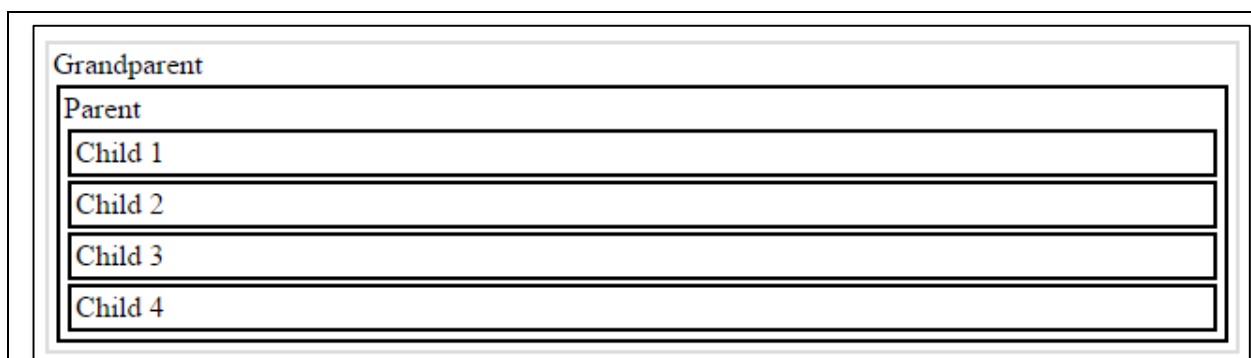


Figure 10.13: The find() Function in JQuery II

10.3.3 Traversing ACCROSS the DOM

The third way the DOM can be traversed is across, where jQuery allows us to navigate to other elements at the same level in the tree. There are several functions to support this.

Traversing Across the DOM	
siblings()	Identifies all other elements with the same parent
next()	Identifies the next element with the same parent
nextAll()	Identifies all subsequent elements with the same parent
nextUntil()	Identifies all subsequent elements with the same parent, until one that satisfies a specified selector
prev()	Identifies the previous element with the same parent
prevAll()	Identifies all previous elements with the same parent
prevUntil()	Identifies all previous elements with the same parent, until one that satisfies a specified selector.

Table 10.2 Traversing Across the DOM

10.3.4 Traversing using filters

Some of the selectors we have used so far can identify many elements on the page, particularly if the page is a large complex one. Filters can be used to narrow down the elements selected. In these examples, the follow page is used, which consists of 4 divs.


```

<html><head>
<style>
  .box { display: block; border: 2px solid #ddd; padding: 2px; margin: 2px; }
</style>
</head><body>
  <div class="box">I am the first</div>
  <div class="box intro">I am the second</div>
  <div class="box">I am the third</div>
  <div class="box">I am the fourth</div>
</body></html>

```

When loaded this page will look as follows.



Figure 10.14: Traversing Using Filters

If we want to choose the first element returned by a selector, we can add the `first()` filter to the selector statement. For example, we could select the divs on the page, and then select only the first div from those returned.

```

<script>
$(document).ready(function() {
  $('div').first().css({"border":"2px solid black"});
});
</script>

```

In this example, only the first div will be affected.



Figure 10.15: The `first()` Function in JQuery

The opposite effect can be gained by using the `last()` filter, which would just return the last element identified by the selector.

```
<script>
$(document).ready(function() {
  $('div').last().css({"border":"2px solid black"});
});
</script>
```

The result would be the fourth and last element being selected.

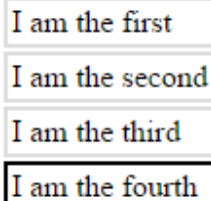


Figure 10.16: The last() Function in JQuery

The eq() filter takes a parameter to specify a number for which element should be selected.

```
<script>
$(document).ready(function() {
  $('div').eq(2).css({"border":"2px solid black"});
});
</script>
```

Here the result would highlight the third div, as counting the elements begins from 0, so the element with index 2 would be the third div.

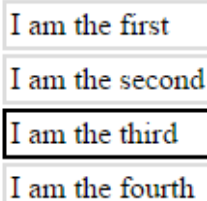
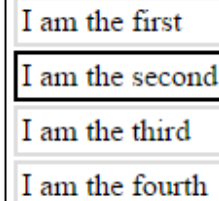


Figure 10.17: The eq() Function in JQuery

Two further filters can be used to test whether an element satisfies a further filter or not.

```
<script>
$(document).ready(function() {
  $('div').filter(".intro").css({"border":"2px solid black"});
});
</script>
```

Here the filter() function is used to reduce the divs to those that also have the intro class, which is our example is only the second div.



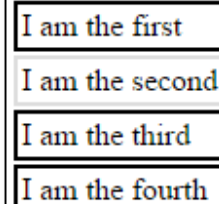
I am the first
I am the second
I am the third
I am the fourth

Figure 10.18: The filter() Function in JQuery

Alternatively we could select elements that don't satisfy an extra condition, by using the not() filter.

```
<script>  
$(document).ready(function() {  
  $('div').not(".intro").css({"border":"2px solid black"});  
});  
</script>
```

This would of course select the other 3 divs.



I am the first
I am the second
I am the third
I am the fourth

Figure 10.19: The not() Function in JQuery

Key Points

- The Document Object Model, or DOM, represents the structure of a webpage with each element on the page an object that are related in a tree structure.
- JQuery offers a variety of functions that make manipulating the DOM straightforward.
- Existing DOM element can be changed, in terms of their text, their HTML, the values and their attributes.
- New elements can be added to the DOM, and elements can be removed.
- The DOM can be navigated, once a selector has been targeted, JQuery makes it easy to move up, down or across the tree to move to another element.

Further Resources

- 1) The JQuery website has a guide to the JQuery functions that are involved in DOM manipulation, which can be found here:-

<https://api.jquery.com/category/manipulation/>

- 2) Tutorialspoint has a collection of samples for how the DOM can be manipulated, that can be found here:-

<https://www.tutorialspoint.com/jquery/jquery-dom.htm>

- 3) A further tutorial offering an introduction to DOM manipulation can be found here:-

<http://www.jquery-tutorial.net/dom-manipulation/introduction/>

Assignment

Now it's time to add to the specification for a web-based calendar app. Similar to Google's calendar, write software that allows a user to keep track of their appointments.

Basic Features:-

- Each user should be able to register and securely login to their calendar.
- Each user should be able to create appointments for particular dates and times.
- Appointments should be stored in a database so the user can view them each time they reopen their browser.
- Users should be able to view appointments in "Day View", "Week View" or "Month View".

The calendar project should be written from scratch in HTML, CSS, PHP, JQuery, MySQL. The features listed here are base features – there are plenty of ways the calendar can "impress"!

Chapter 11

Asynchronous JavaScript and JQuery UI

Objectives

This chapter focuses on more advanced features of JQuery, beginning with how to use it for asynchronous web applications where different parts of the page are loaded at different times using a technique commonly referred to as AJAX. The chapter also looks at the JQuery UI extension which adds further user interface features to the core library, such as dialog boxes and the ability to drag elements around the screen. After reading it you should:-

- Understand how an AJAX call can be used to update a page.
- Be familiar with the key AJAX functions in the JQuery library - load(), get(), post() and ajax().
- Be able to add further User Interface features including date pickers, dialog boxes and enable the page visitor to drag elements around the screen.

Contents

11.1 Introducing AJAX

11.2 JQuery and AJAX

11.3 JQuery

The previous two chapters have introduced JQuery, demonstrating how the library can be used to make writing JavaScript easier. The library contains a wide variety of JavaScript functions that can be used to handle events such as when a page visitor interacts with elements on the page, and also some functions for manipulating the DOM. This chapter begins by using JQuery to communicate with the server, and then looks at JQuery extensions that can be used to make improve the user interface.

11.1 Introducing AJAX

AJAX stands for Asynchronous JavaScript and XML and is used for making asynchronous web applications, which means that different parts of the page can be loaded at different times. Early websites were built on the principle that when users interacted with a site, a complete new HTML file needed to be requested and sent from the server. If the interaction only required changing a small part of the page, this becomes very inefficient and places unnecessary loads on the server, and bandwidth, it also results in a poor user experience.

AJAX stands for Asynchronous JavaScript and XML and is used for making asynchronous web applications, which means that different parts of the page can be loaded at different times.

One early approach to resolving this was to divide the page into frames, whereby each section of the page could load a different HTML file and could update separately. While this approach was popular for a while, it has several drawbacks and is no longer supported in HTML5. Some of the problems with frames include difficulties for search engines to properly index pages, different parts of the page appearing differently on different screens due to different resolutions, the back button not working as expected and some devices, particularly smaller devices not being big enough to be divided up. Needless to say, there is a better way.

As well as being able to update parts of a page independently from each other, AJAX provides a convenient way for the browser to communicate with the server, allowing the webpage to request, and receive, data from a server after it has been loaded, as well as send data to the server in the background. AJAX is not a programming language itself, it is just a combination of some JavaScript running in the browser, and using the browsers XMLHttpRequest object to request data from the server. It is called AJAX because originally XML (eXtensible Markup Language) was used to transport the data, but today is it common to transport the data as plain text, or in JSON (JavaScript Object Notation).

Figure 11.1 shows the general way that AJAX works. We have previously seen different events that can occur on a webpage, such as clicking on a button, or when a page loads. Here, when an event occurs an XMLHttpRequest object is created by JavaScript. This object is then sent to the server, which processes it and creates a response to send back to the browser. The response could be in XML form, or JSON for example. When the response arrives, the browser uses JavaScript to respond appropriately, such as by updating the page with the new data.

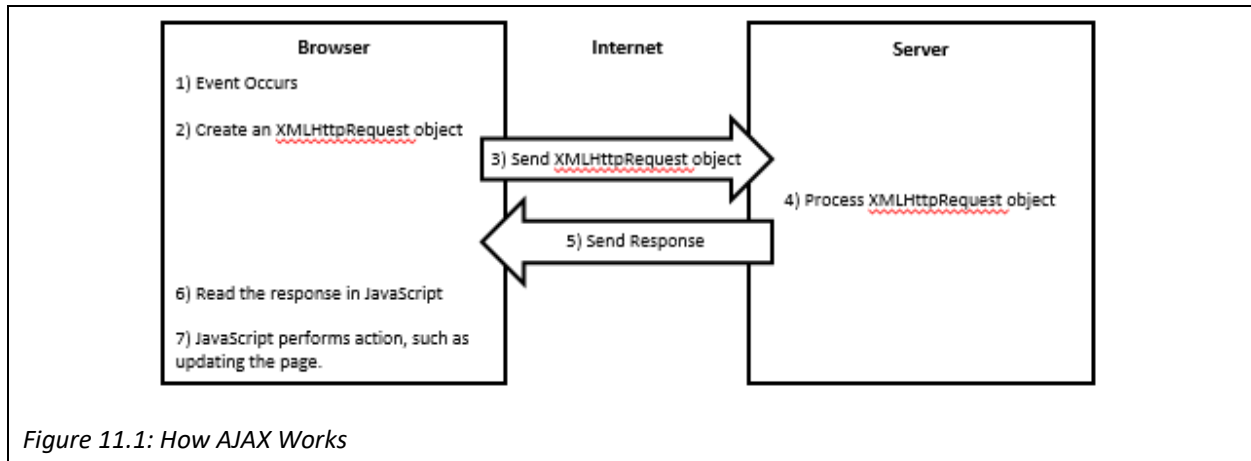


Figure 11.1: How AJAX Works

Clearly the XMLHttpRequest object is an important part of this process, and fortunately the object is a built in component of all modern browsers (Chrome, Firefox, Opera, IE 7+, Edge, Safari). Unfortunately as a developer you can't control which browser is being used, and older browsers may not be compatible – IE5 and IE6 use an ActiveObject instead of the XMLHttpRequest object. Fortunately, as we will soon see, JQuery again includes methods that make AJAX much simpler and function across browsers. Before investigating JQuery and AJAX it is worth looking at how an AJAX request might be written in raw JavaScript.

```
<html>
<body>
<p id="demo" onclick="UpdateContent()">Original Page Content.</p>
<script>
function UpdateContent() {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xmlhttp.open("GET", "newdata.txt", true);
  xmlhttp.send();
}
</script>
</body>
</html>
```

The page is a simple example of an AJAX function call. Firstly the page consists of a single paragraph element, which calls the UpdateContent() function when it is clicked upon. This function creates a new XMLHttpRequest object and then defines a function to be called whenever the readyState property changes for it. The readyState property will change several times during a Http request, as it is sent to the server, with ultimately the 4th readyState occurring when the request has been completed and a response returned.

0	Request not initialized
1	Server connection established
2	Request received
3	Processing request
4	Request finished and response is ready

Table 11.1: The readyState Property of XMLHttpRequest

Once the completed readyState is reached, and the status is “OK” (or 200), the text in the paragraph is changed to whatever the server responds with, converted to a string. Before the request is sent to the server, the url is specified. For security reasons the file must be located on same domain, and in this example the requested file is newdata.txt, as indicated in the following line. To test the code, just create a text document with any content, when you click on the paragraph, the text will change to the contents of newdata.txt.

```
xhttp.open("GET", "newdata.txt", true);
```

In this line of code, the third parameter is set to ‘true’, to indicate that the request should be done asynchronously. Server requests should be asynchronous, so that JavaScript can continue executing other scripts while waiting for the server to respond, and deal with the response only when it is ready. AJAX is a powerful tool that is found throughout the web, but as can be seen from the sample above, coding it can get tricky, particularly when needing to deal with different browsers. Fortunately JQuery contains some AJAX methods that make AJAX much simpler.

11.2 JQuery and AJAX

The JQuery library contains several functions to support AJAX, allowing the browser to load fresh content from the server, or post data back to the server, along with callback functions to react after the server responds.

11.2.1 JQuery load() Method

The load() method is the simplest ajax function, allowing data to be called from a server and returned into a selected element. At its simplest, it could load the contents of a text file into a specified element, such as;

```
$("#div1").load("data.txt");
```

In this simple example, the contents of data.txt will be displayed in the element with id ‘div1’. Clearly the data to be loaded need not be a text file, it could be a php file that includes queries to your database, which could be used to update the latest content. The file could also contain standard html, in which case a specific selector could be specified to target a particular part of the file. In the following example, the contents of the element with id “news” is loaded from data.htm, and inserted into div1.

```
$("#div1").load("data.htm #news");
```

Given that the function requires a response from the server, which for many reasons will take an unpredictable amount of time, a callback function can be added, which can be particularly useful to perform

some task after the response has been returned. The callback function has a few different parameters, containing the information returned from the server;

responseTxt – contains the content returned assuming the call was a success.

statusTxt – contains the status of the call, for example “success” or “error”.

xhr – contains the actual XMLHttpRequest object.

The callback function can therefore be used to react appropriately in case the server returns an error. In the following example, the contents of data.txt is loaded into div1, however, for testing purposes a callback function is used to write an error to the console if the server returns an error message.

```
$("#p1").click(function(){
  $("#div1").load("data.txt", function(responseTxt, statusTxt, xhr){
    if(statusTxt == "error") {
      console.log("Error" + xhr.status + " " + xhr.statusText);
    }
    if(statusTxt == "success") {
      console.log("Successfully Loaded New Data");
    }
  });
});
```

11.2.2 JQuery \$.get() Method

The \$.get() method requests data from the server using the HTTP GET request, with the returned data being available in a callback function when the server successfully responds. Unlike the load() method, the data isn't immediately loaded into an element, instead the callback function determines what to do with the data. The function takes 2 parameters, as shown below, the first being the URL being requested on the server, and the second being the callback function.

```
$("#p1").click(function(){
  $.get("data.txt", function(data, status){
    console.log(data + ": status : " + status );
  });
});
```

11.2.3 JQuery \$.post() Method

The \$.post() also requests data from the server, but this time using the HTTP POST request, so it is often used to send some data to the server for processing. This time the function takes 3 parameters, with the

data being posted to the server being the second parameter. In this example, a name and id is sent to the `add_student.php` file on the server. The `add_student.php` file could then insert that data into a database.

```
$("#p1").click(function(){
  $.post("add_student.php",
  {
    name: "Ken",
    id: "1234"
  },
  function(data, status){
    console.log(data + ": status : " + status );
  });
});
```

11.2.3 JQuery `$.ajax()` Method

The AJAX methods introduced so far all invoke a further `$.ajax()` method, which generally only needs to be used if the `.load()`, `$.get()` or `$.post()` methods aren't sufficient. The `$.ajax()` method sends a set of name/value pairs to the server, allowing more versatility to the call, for example an AJAX call is asynchronous by default, but it is possible to set a synchronous request by setting `"async: false"`. Clearly making the request synchronous may lock the browser until it completes. A username and password could be passed securely using the `$.ajax()` method. The code example below performs much the same as in the `$.post()` method described above, and in general the `$.post()` method is preferred; the following example is only intended to demonstrate the syntax of an `$.ajax()` call.

```
$("#p1").click(function(){
  $.ajax({
    url:"add_student.php",
    data: { name: "Ken", id: "1234" },
    method: POST })
  .done(function(html){
    console.log(html);
  });
});
```

11.3 JQuery UI

The previous chapters have demonstrated how JQuery makes JavaScript easier, and helps make webpages come to life. JQuery UI has been built on top of JQuery supporting a variety of user interface effects, interactions and widgets designed to help make websites more professional, and more appealing. Some examples of the features of JQuery UI include attractive date pickers, the ability for the user to drag and drop elements around the page, menus and dialog boxes.

11.3.1 Getting Started with JQuery UI

Just like the main JQuery library, JQuery UI can be downloaded and installed on your server, or accessed from the JQuery website. If you choose to host it yourself, during the download process you will be able to customize the css theme so that it matches your page. Three lines of code need to be added to the header of your webpage to link to the JQuery UI css and js files.

```
<link rel="stylesheet" href="jquery-ui.min.css">
<script src="external/jquery/jquery.js"></script>
<script src="jquery-ui.min.js"></script>
```

After the js and css files are included, any of the features of JQuery UI can be accessed.

11.3.2 A Simple Date Picker

Because dates can take many different formats some care is needed when asking a user to input a date - most notably the difference between dd/mm/yyyy vs mm/dd/yyyy. Fortunately with HTML5 the input type of date provides an easy way of giving a drop down date picker. Later chapters will explore HTML5 features further, but the date picker is browser dependent, meaning it may appear differently in different browsers and may not be available for older browsers.

```
<input type="date">
```

The standard HTML input type date can be used on forms to get a date input.

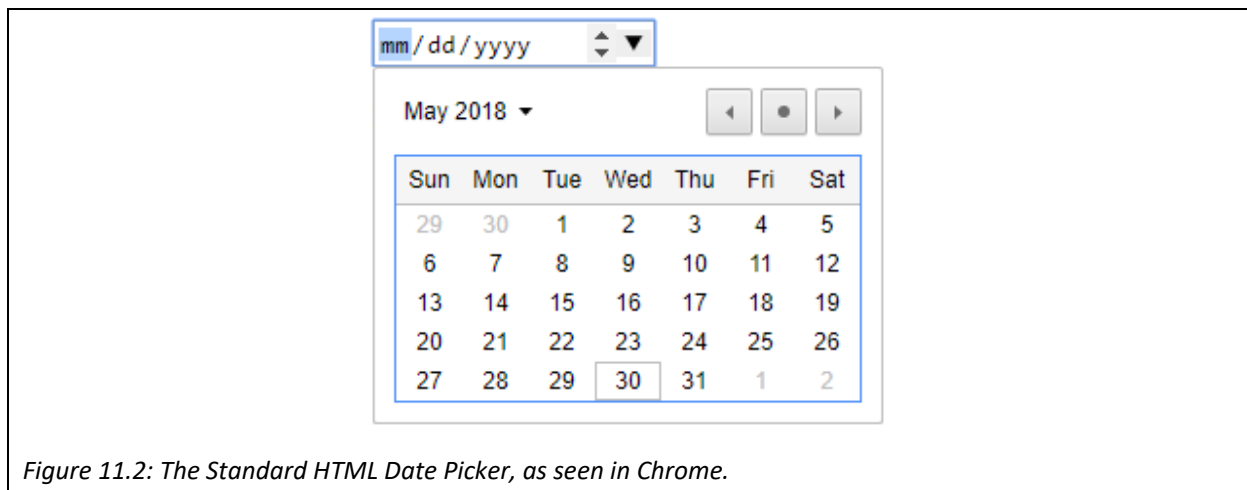


Figure 11.2: The Standard HTML Date Picker, as seen in Chrome.

The current version of Chrome displays a dropdown calendar format as seen in Figure 11.2, while in Edge the same code displays a dropdown set of sliders for the month, day and year as seen in Figure 11.3.



Figure 11.3: The Standard HTML Date Picker, as seen in Edge.

One way of creating a consistent view is to use the JQuery UI date picker. To do so involves adding an id to a text input box, and then adding JavaScript code to add the datepicker function to the id.

```

<script>
$( function() {
  $("#date").datepicker();
});
</script>
Date: <input type="text" id="date">

```

The result can be seen in Figure 11.4, with the appearance being consistently controllable across all browsers.

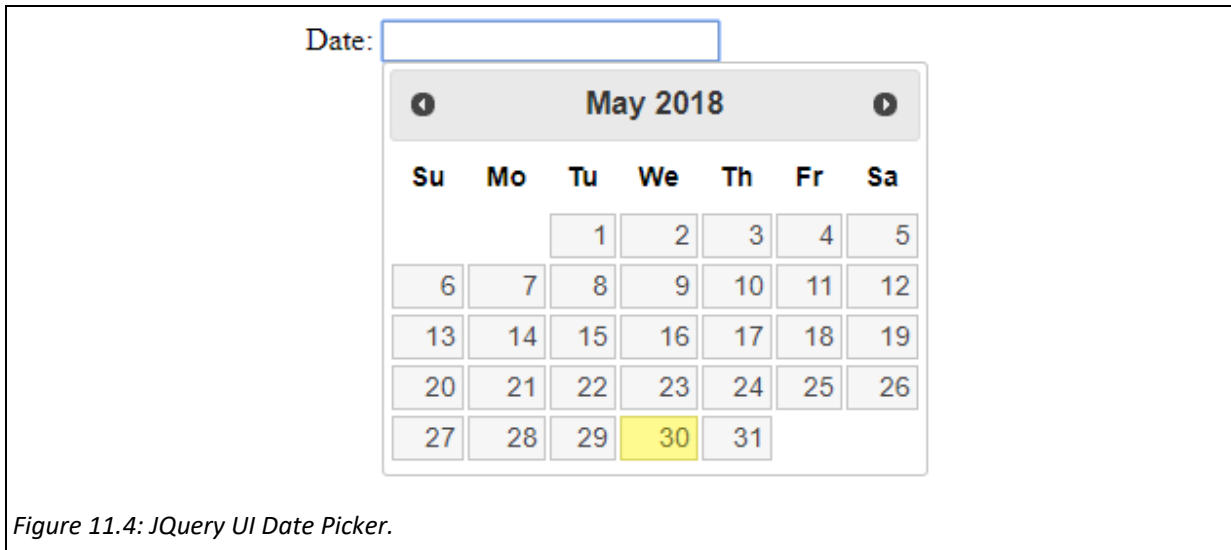


Figure 11.4: JQuery UI Date Picker.

11.3.3 Dialog Boxes

A Dialog Box, sometimes called a Modal Box, is a window that appears over the page like a popup box, which can display more information to the user, or collect information from them. JavaScript has several popup boxes, such as the alert box previously used during testing. Dialog boxes can be a useful part of the user interface design, so being able to design attractive modals contributes to the user experience.

```
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<link rel="stylesheet" href="ui/jquery-ui.min.css">
<script src="https://code.jquery.com/jquery-1.12.4.js"></script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
<script>
$(document).ready(function() {
  $("#appointment").click(function() {
    $("#dialog").dialog();
  });
});
</script>
</head>
<body>
<div id="appointment">Click here for details</div>
<div id="dialog" style="display:none">
<p>Web Programming Final Exam at 12:00pm</p>
</div>
</body>
</html>
```

The example on the previous page uses JQuery UI to create a very simple dialogbox. To begin with the page contains two divs – one that is visible, and a second that is invisible. When the user clicks on the first div, the contents of the second div appear in an attractive dialog box. The code for creating the dialog box is a simple call to the dialog() function when the appointment div is clicked. The result is a box that the user can move around and resize as seen in Figure 11.5.

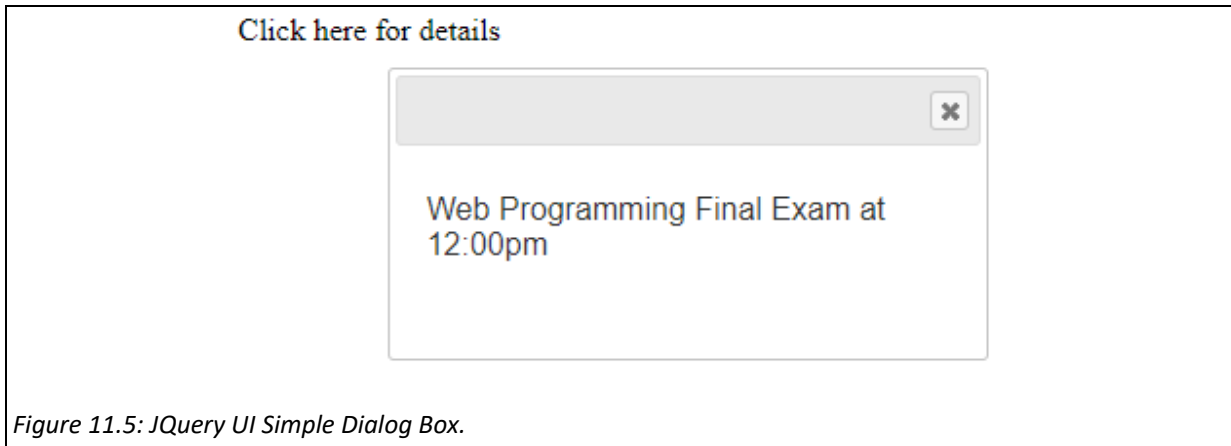


Figure 11.5: JQuery UI Simple Dialog Box.

The example given here is a very simple dialog box, where only some fixed text is displayed. Using the techniques discussed previously in this chapter, it isn't difficult to change the contents of the dialog box, including querying the database on the server to find the correct appointment information, or creating a form where the user could edit the appointment information before posting new content to the server. By using a dialog box, the user need not reload the page when creating new appointments and submitting them to the database.

11.3.4 Draggables and Droppables

Another cool, yet simple, feature of the JQuery UI library is the ability to allow the user to move elements around the screen using their mouse. Throughout this book a calendar has been developed, similar to the google calendar. One of the key features of that interface is being able to move appointments from one day to another, or from one timeslot to another. This feature can be easily added using the JQuery UI draggables and droppables feature.

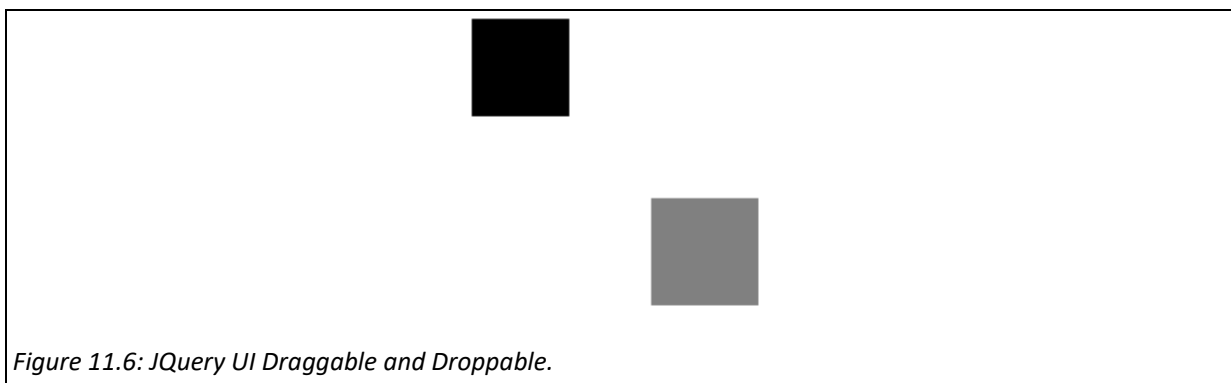


Figure 11.6: JQuery UI Draggable and Droppable.

Any element in the browser can call the `draggable()` function and then be moved around the window – in this example the black box is given the draggable ability. Other elements can call the `droppable()` function, and then a further function be called when draggable elements are dropped on them – in this example the grey box is given the droppable ability. Figure 11.6 shows the original layout, and in the following code an alert is given when the black box is dropped onto the grey box. Of course, the drop event could call any function involving any relevant updates.

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="jquery-ui.css">
<script src="jquery.min.js"></script>
<script type="text/javascript" src="jquery-ui.min.js"></script>
<style type="text/css">
  #draggable { width: 50px; height: 50px; background: black;}
  #droppable { position: absolute; left: 100px; top: 100px; width: 55px; height: 55px; background: gray;}
</style>
<script>
$(document).ready(function() {
  $("#draggable").draggable();
  $("#droppable").droppable({
    drop: function() { alert('dropped'); }
  });
});
</script>
</head>
<body>
<div id="droppable"></div>
<div id="draggable"></div>
</body>
</html>
```

There are plenty more features included in the JQuery UI library, such as an attractive way of sorting a list of elements by dragging and dropping them into position. The features discussed here should enable you to add some further features to your calendar.

Key Points

- Asynchronous JavaScript allows the browser to get updates from the server and change parts of the page independently.
- An XMLHttpRequest object is used by the browser to send a request to the server.
- JQuery makes AJAX simpler using its `load()`, `get()` and `post()` methods.
- If those methods aren't enough, the versatile `ajax()` method can be used.

- The JQuery UI extension can be used to improve the user interface, it includes several features including a date picker, dialog boxes, draggables and droppables.

Further Resources

- 1) The w3schools website has an introduction to JavaScript's ajax, which can be found here:-
https://www.w3schools.com/js/js_ajax_intro.asp
- 2) The JQuery website has a detailed description of all of its methods, including the ajax() method, which can be found here:-
<http://api.jquery.com/jquery.ajax/>
- 3) JQuery UI can be downloaded from the following link, along with full documentation of all its features.
<https://jqueryui.com/>

Assignment

It's time to add to the specification for a web-based calendar app. Similar to Google's calendar, write software that allows a user to keep track of their appointments.

New Features:-

- Users should be able to create appointments in a modal, (without reloading their webpage).
- Users should be able to edit and delete appointments in a modal, (without reloading their webpage).
- Users should be able to change an appointment by dragging it to a new time (or day).
- Users should be able to change the length of an appointment by dragging it.

Part 4

Modern Web Development Technologies

Chapter 12

HTML 5

Chapter 15

Express.js

Chapter 13

Introducing Node.js

Chapter 16

Angular

Chapter 14

Node.js, MySQL & MongoDB

Web development is a fast-paced environment with new technologies emerging regularly – new devices are connecting with new interfaces and more of those devices are mobile devices leading to new challenges, including the challenge of Big Data. The earlier parts of this book have focused on traditional, well established approaches to web development, using the *AMP stack. While much of the existing web has been developed using those technologies, much new development is done using new techniques and capabilities. This section introduces some of the features of the latest version of HTML – HTML5 and then explores the MEAN development stack consisting of MongoDB, Express.js, Angular and Node.js. JavaScript is a powerful language which is now powering both the client and the server.

Chapter 12

HTML 5

Objectives

This chapter introduces the new features that were added to HTML for version 5. In October 2014, the World Wide Web Consortium (W3C) released the latest version of HTML, known as HTML5. The new release includes new tags, and new capabilities, some of which are described in this chapter. While not all features of HTML5 are compatible with all browsers, particularly users using outdated browsers, the capabilities described here demonstrate the future of the web. After reading it you should:-

- Be familiar with the new Semantic Elements included in HTML5
- Understand how rich audio and video media should be included on a webpage
- Be able to draw simple shapes onto the Canvas using JavaScript code
- Be familiar with the new capabilities the Geolocation API offers
- Understand how to use localStorage to store data in the browser
- Be able to use a Web Worker to perform CPU intensive tasks in the background without interrupting the user interface

Contents

- 12.1 New Tags (and Deprecated Tags)
- 12.2 Audio & Video
- 12.3 The Canvas
- 12.4 Geolocation
- 12.5 Local Storage
- 12.6 Web Workers

HTML was originally defined in 1991, by Tim Berners-Lee, but clearly the world of the web has massively changed since the early years. While early websites were plain and largely text based, today web programs are rich in varied media from videos to games. Early websites were displayed on primitive machines, while now we access the web using a variety of different types of powerful devices from laptops to mobile phones and tablets. Clearly HTML remains a fundamental language and so in October 2014 the World Wide Web Consortium (W3C) released the latest version of HTML, known as HTML5. All modern browsers support HTML5, but while the language is specified by W3C, some features are yet to be adopted by some browsers, so even several years later there are some compatibility issues to be aware of. This chapter will investigate some of the key features of HTML5.

12.1 New Tags (and Deprecated Tags)

Because there have been many different versions of HTML, it is important to instruct the browser which type of document it is displaying, which is done with a doctype declaration – which needs to be the first line of your page, before the opening <html> tags. Previously the doctype declaration was more complicated, but with HTML5, this is much simpler.

```
<!DOCTYPE html>
```

Technically this is not new and also not a tag, it is an instruction to the browser so it knows how to render the page, strictly following the specification to ensure the page is displayed how the developer intended it to. It might be tempting to forget to add the doctype declaration, as it may appear to make no difference to how the page appears in your browser, however a different browser version may switch to quirks mode (rather than standards mode) and may not be able to display your page exactly as intended.

Also at the beginning of a document, the character encoding (or charset) of that document should be specified. Again. Over the years different character encodings have been developed, from ASCII, ISO-8859-1, ANSI to UTF-8. Previously this was specified by a complex meta tag, which in HTML5 has again been simplified. UTF-8 is the default character encoding for HTML5, and should be declared with the following command immediately after the <head> tag is opened.

```
<meta charset="UTF-8">
```

Therefore the standard structure to a page in HTML5 is as follows.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Page Title</title>
</head>
<body>
</body>
</html>
```

12.1.1 New Semantic Elements

Over recent decades there has been much discussion about developing a semantic web, where data is structured so that machines can directly read it. HTML5 has introduced some new semantic elements or tags to better describe the different parts of a webpage. Semantic elements are those that convey some meaning both to a browser and a developer, for example the <form> tag clearly defines what is inside it, whereas the <div> tag says nothing about its contents. Traditionally developers identify parts of their page using tags such as <div class="header"> or <div class="nav">, HTML5 has created consistency with some new semantic tags, such as <header> and <nav>. These semantic tags can be styled using CSS much like elements that have a class selector, the difference is that the intention behind semantic elements is to make it easier for a machine to understand the different parts of a page, enabling it to reuse data across different applications.

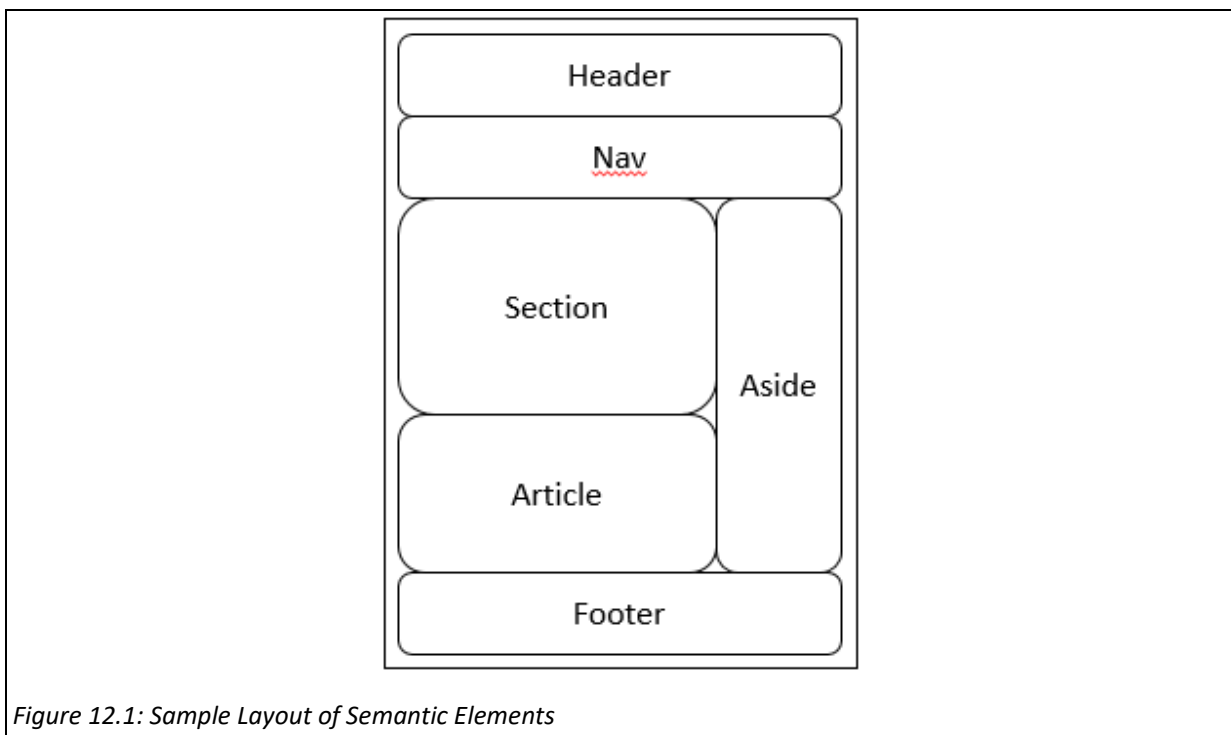


Figure 12.1: Sample Layout of Semantic Elements

Figure 12.1 shows a sample layout of semantic elements with some of the key tags showed. Since they are designed to be semantic tags, most should be self-explanatory, nonetheless table 12.1 shows a list of the new semantic tags along with a description. Note the relationships between some of these elements, for example a <summary> element contains a brief header to a <details> element, while a <figcaption> element provides the caption for a <figure> element. There is also a relationship between the <article> element and the <section> element, as a section may contain several articles, whilst an article may contain several sections. The <header> and <footer> elements may refer to the header (or footer) of an entire page, or just to the opening and closing of a particular article or section. While this might seem confusing, the intention is that the new tags add more meaning to the content than simply <div> or tags. As these tags are relatively new, it is worth monitoring the source of some of your favourite websites to see how they are adopted.

Tag	Description
<article>	For articles
<aside>	Other content apart from the page content
<details>	Additional details that can be hidden from a <summary> element
<figcaption>	A caption for a <figure> element
<figure>	A figure (diagram, photo, illustration etc.)
<footer>	The closing section of a page (or section)
<header>	The opening section of a page (or section)
<main>	For specifying the main part of a document
<mark>	For highlighting part of the text
<nav>	For navigation links
<section>	For a section in the content
<summary>	A visible heading for a <details> element
<time>	For defining dates or times

Table 12.1 HTML5 Semantic Elements

12.1.2 New Input Types

Prior to HTML5, there were several input types available for form input, including text, password, radio (buttons), checkboxes, and then submit and reset buttons. HTML5 adds to this list with further input types, which can make form validation easier, by trapping input errors, and also improve the user experience. For each of the following input types, if they are not supported by the browser, they will simply be replaced with a text box input.

```
<input type="color" value = "#bbbbbb">
```

The color input type shows a palette for the user to choose from as in Figure 12.2.

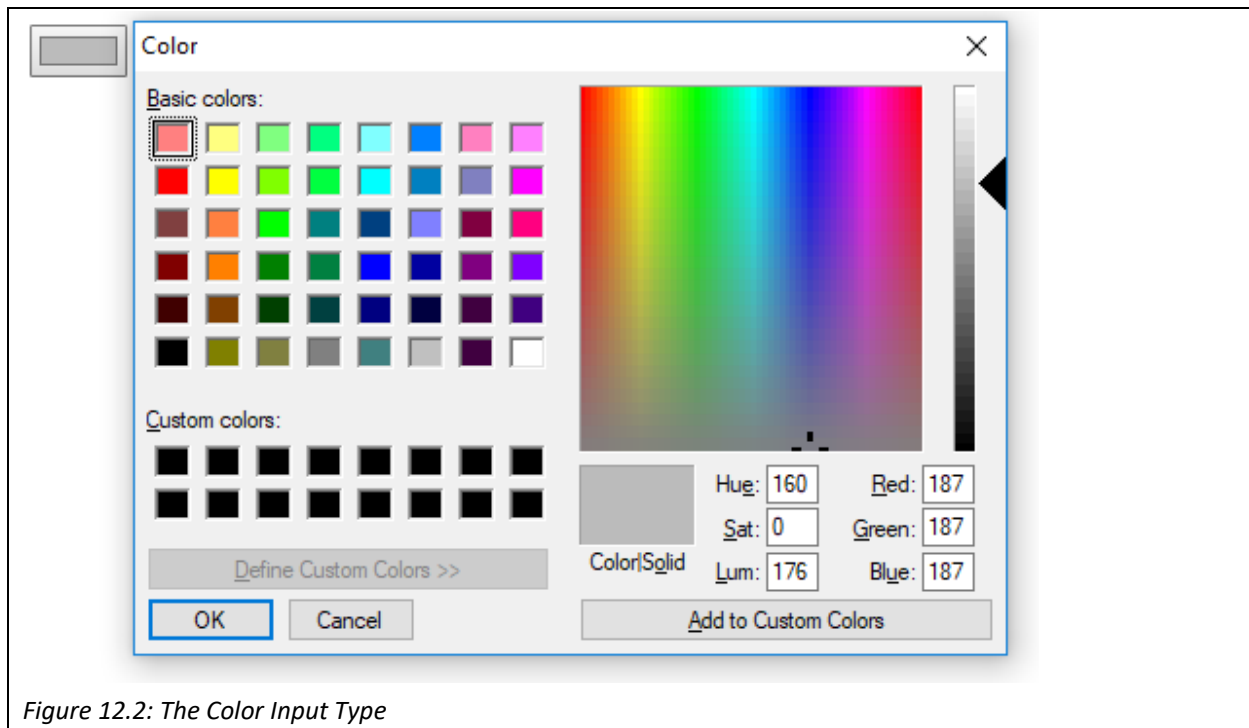


Figure 12.2: The Color Input Type

The previous chapter introduced the date input type with a drop down menu to choose from and discussed the HTML5 version. A further HTML5 feature for several input types is to allow min and max attributes to restrict the range of choices available.

```
<input type="date" min="1950-01-01" max="2018-12-31">
```

Related input types include “datetime-local”, which allows the time to be input as well as the date, “month”, which allows the user to select a month and year and “week”, which allows the user to select a week and year. The “email” and “url” input types have been improved to ensure a valid input is given, for example an email should contain an “@”, and an appropriate error message is given when the input doesn’t conform. Finally the “range” input type allows the user to input their score using a slider control.

```
<input type="range" name="score" min="0" max="10">
```

While features like this could be implemented using JavaScript, HTML5 includes them as part of their specification.



Figure 12.3: The Range Input Type

Further new HTML5 tags have been introduced for dealing with rich media, such as <audio>, <video> and the <canvas>, but these will be discussed in more detail in the following sections.

12.1.3 Deprecated Tags

The support for some tags has been removed in HTML5, largely as they have been replaced by better methods. To begin with the <frame>, <frameset> and <noframes> tags have been removed, as discussed in the previous chapter, AJAX provides a much superior way of controlling different parts of the page asynchronously. Also, several tags that are just concerned with style have been removed, as CSS is the standard way of managing the style of a page. These tags include <basefont>, <big>, <center>, , <strike> and <tt>, so clearly these tags should be avoided as future browsers will not be able to display them.

12.2 Audio & Video

Prior to HTML5, embedding audio and video into websites wasn’t ideal as it generally involved using a plugin, like Adobe’s Flash. This meant that the browser wasn’t able to play the content itself, instead users needed to download a proprietary player, which while free to download made the experience not seamless. Creating websites using Flash was for a while an alternative to the HTML, CSS, JavaScript methods discussed in this book, and in many ways it was easier to develop flashy looking websites using Flash, however the tools needed were expensive. HTML5 has sometimes been touted as the death of Flash, and the popularity of open tools has resulted in the gradual removal of Flash from the web. Other alternatives include some JavaScript media players, but the introduction of the HTML5 <audio> and <video> tags have made integrating rich media into websites much easier.

The <audio> tag makes it easy to play music and other sounds directly from the browser and is supported by most modern browsers. There are a couple of key parts to adding audio to a website, first adding

the controls attribute to the audio tag adds buttons like play and pause, secondly the source of the audio needs to be specified, depending on what type of audio it is.

```
<audio controls>
<source src="music.mp3" type="audio/mpeg">
<source src="music.ogg" type="audio/ogg">
Audio Not Supported.
</audio>
```

Unfortunately not all browsers support the same type of audio formats, so often more than one audio file needs to be specified in different formats, however MP3 appears to be better supported than wav or ogg. Adding controls to the audio tag results in the appearance such as in Figure 12.4.



Figure 12.4: Audio Playback Controls

The <video> tag works in a similar way, except that it is always a good idea to specify the height and width parameters of the video window. There is also the option of adding the autoplay attribute to make the video start straightaway.

```
<video width="320" height="240" controls autoplay>
<source src="movie.mp4" type="audio/mp4">
<source src="movie.ogg" type="audio/ogg">
Video Not Supported.
</audio>
```

Once again, not all video formats are supported by all browsers, with the choices being MP4, WebM and ogg, with MP4 being the best supported. The controls for video playback are similar to audio as can be seen in Figure 12.5.

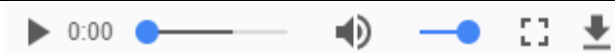


Figure 12.5: Video Playback Controls

Both the <audio> and <video> tags interact with the DOM, allowing them to be edited using JavaScript, for example using the play() or pause() methods to start or pause the media.

12.3 The Canvas

The <canvas> tag provides a space where graphics can be drawn onto a page. JavaScript can then be used to dynamically draw onto the canvas a variety of shapes and effects. Rather than placing a static image onto a page, the canvas allows designers to draw graphs, graphics, games and other visuals, on the fly, in real time. The canvas therefore can be used to create interesting ways of presenting data or advertising, and with the JavaScript interactions animations and games can also be built. Prior to HTML5, again this was the domain

of Flash, where users would need to download external software to play the Flash animation, by using the canvas users can access such animations directly in their browser.

The <canvas> itself is just a blank space, and JavaScript can then be used to draw onto the space. This section will therefore only introduce some very basic examples of its use with interested readers encouraged to investigate further the use of the canvas for building games and other rich multimedia. It is worth noting that another tag <svg> is a container for creating scalable vector graphics which may be an alternative. The simplest <canvas> tag involves setting the width and height of it, in this example a border is also set, while any text inside the canvas element will only be displayed on browsers that don't support HTML5.

```
<canvas id="canvas1" width="400" height="250" style="border:1px black solid">  
  This is a canvas element, you need HTML5!  
</canvas>
```

This would display a plain empty canvas with height 250 pixels, width 400 pixels, and a border around it. The next step is to use JavaScript to draw onto it.

```
<script>  
$(document).ready(function(){  
  canvas = $('#canvas1')[0];  
  context = canvas.getContext('2d');  
  context.moveTo(20,50);  
  context.lineTo(380,150);  
  context.stroke();  
});  
</script>
```

In this example, first the canvas element is selected, and then the context set to 2 dimensional, rather than 3 dimensional. A line is then drawn from pixel (20,50) to pixel (380,150), resulting in Figure 12.6.

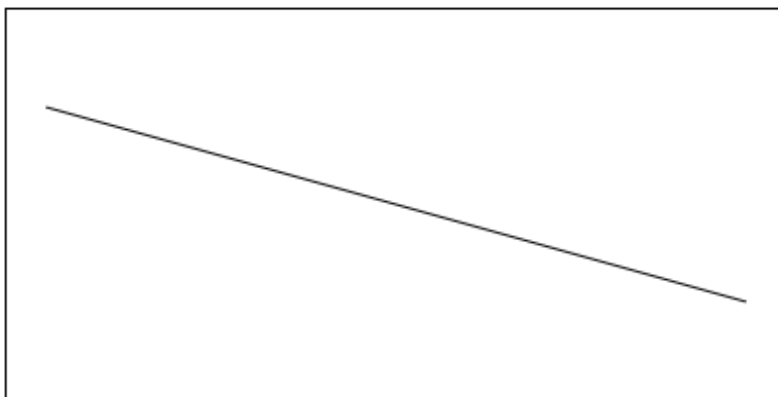


Figure 12.6: Simple Line Drawn onto Canvas

As well as the `moveTo()`, `lineTo()` and `stroke()` methods, the canvas, via JavaScript, has plenty of methods allowing the creation of any variety of shapes using code, including circles, as in this second example,

where replacing the last 3 lines of the previous example as follows, draws a red circle in the middle of the canvas like the Japanese Flag shown in Figure 12.7.

```
context.fillStyle = 'red';
context.beginPath();
context.moveTo(200, 125);
context.arc(200, 125, 70, 0, Math.PI * 2, false);
context.closePath();
context.fill();
```

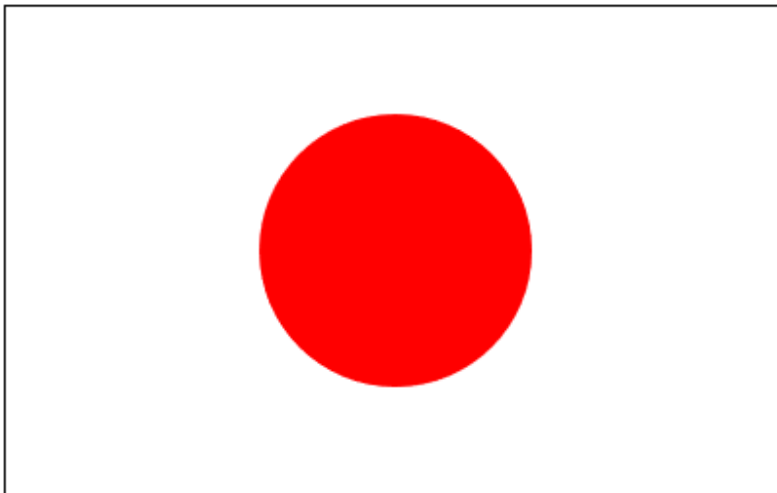
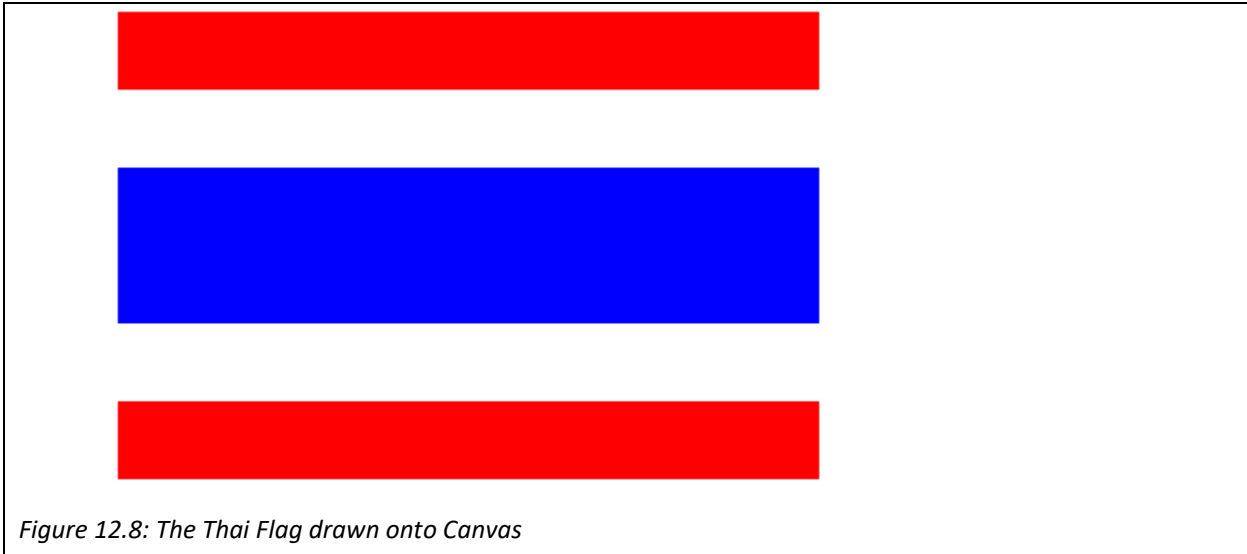


Figure 12.7: The Japanese Flag drawn onto Canvas

There are a variety of ways to draw a Thai Flag, one of which would be as follows.

```
$(canvas).css("background", "white");
context.fillStyle = 'red';
context.fillRect(0, 0, 400, 250);
context.clearRect(0, 42, 400, 168);
context.fillStyle = 'blue';
context.fillRect(0, 84, 400, 84);
```

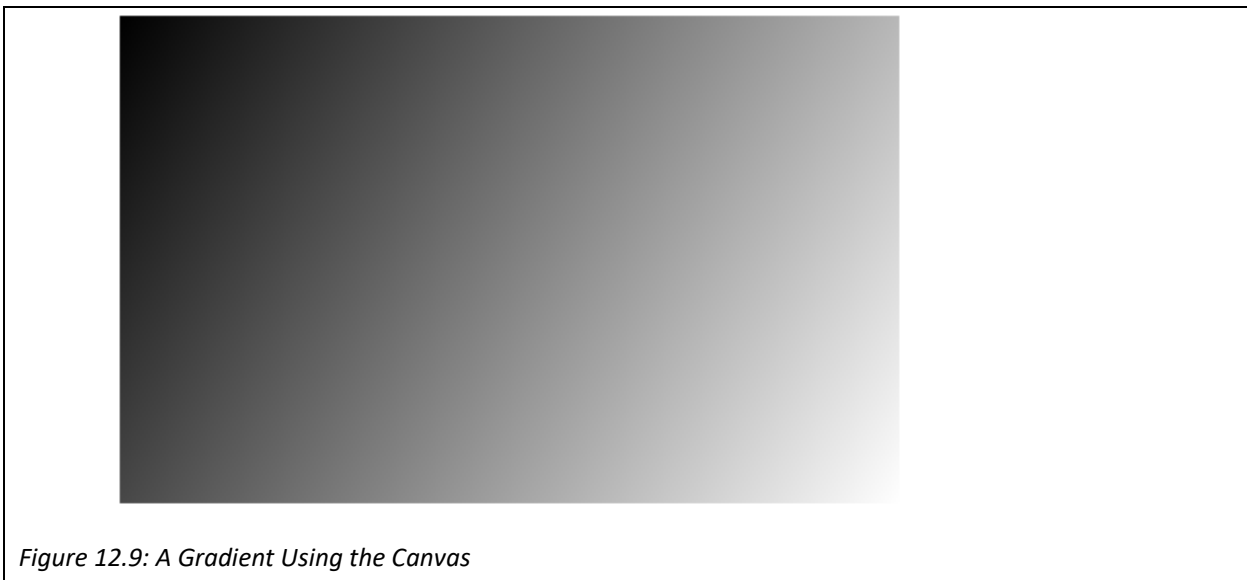
In this example, the full space is filled red, before a white rectangle cleared in the middle. Finally the blue central band is placed on top.



In these examples the shapes were filled with solid colour, but it is easy to create a gradient between the colours as in the next example where the colour shades from black to white, from the top left corner to the bottom right corner.

```
gradient = context.createLinearGradient(0, 0, 400, 250);  
gradient.addColorStop(0, 'black');  
gradient.addColorStop(1, 'white');  
context.fillStyle = gradient;  
context.fillRect(0, 0, 400, 250);
```

This results in the graphic in Figure 12.9.



This section has just introduced some of the basics of the HTML5 canvas, as it has much potential beyond the scope of this book. At very least it intends to become an alternative to Flash for rich graphics and multimedia.

12.4 Geolocation

Given that more and more users access the web via mobile devices, support for mobile services was an essential feature added into HTML5. Therefore the Geolocation API is there to get the longitude and latitude of a user. Clearly there are privacy and security issues with using geolocation, so the user's permission is needed before accessing it. Assuming permission is granted, geolocation can be linked with mapping software, such as Google Maps, and location dependent services offered. In this example, the user's location is requested and a map displayed.

```
<!DOCTYPE html>
<html><head>
  <meta charset="UTF-8">
  <title>Geolocation</title>
  <script src="jquery-1.11.1.min.js"></script>
  <script src="https://maps.googleapis.com/maps/api/js?sensor=false"></script>
</head><body>
  <div id='status'></div>
  <div id='map'></div>
  <script>
$(document).ready(function(){
  if (typeof navigator.geolocation == 'undefined')
    alert("Geolocation not supported.")
  else
    navigator.geolocation.getCurrentPosition(granted, denied)
  function granted(position) {
    $('#status').innerHTML = 'Permission Granted'
    $('#map').css("border", "1px solid black")
    $('#map').css("width", "640px")
    $('#map').css("height", "320px")
    var lat = position.coords.latitude
    var long = position.coords.longitude
    var gmap = $('#map')[0]
    var gopts = {
      center: new google.maps.LatLng(lat, long),
      zoom: 9, mapTypeId: google.maps.MapTypeId.ROADMAP }
    var map = new google.maps.Map(gmap, gopts)
  }
  function denied(error) {
```

```

var message
switch(error.code) {
  case 1: message = 'Permission Denied'; break;
  case 2: message = 'Position Unavailable'; break;
  case 3: message = 'Operation Timed Out'; break;
  case 4: message = 'Unknown Error'; break;
}
$('#status').innerHTML = message
}
});
</script>
</body>
</html>

```

This fairly standard example requests the users permission, and if granted displays a Google Map of their location, as in Figure 12.10. The Google Maps API offers a much larger range of features such as placing a pin on the user's precise location, or highlighting nearby attractions.



Figure 12.10: Google Map Powered By Geolocation

12.5 Local Storage

HTML5 has also transformed cookies, allowing more data to be stored more securely on the client machine within the browser. The data storage limit is higher, and data isn't transferred to the server. Local storage can either store data indefinitely or for the duration of a session depending on whether it uses window.localStorage or window.sessionStorage.

```

<div id='set'>Set</div>
<div id='unset'>Unset</div>
<script>
$(document).ready(function(){
$("#set").click(function() {
    localStorage.setItem('username', 'ken')
    localStorage.setItem('password', 'pass')
});
$("#unset").click(function() {
    localStorage.removeItem('username')
    localStorage.removeItem('password')
});
});
</script>

```

In this example a username and password are stored locally in the localStorage, which can be viewed through the browser's developer tools. If the user clicks on "Set" the data is stored, and when they click on "Unset" the data is removed. Figure 12.11 shows a screenshot of the developer tools after "Set" has been clicked. Note that for security reasons this demo needs to be run on a server, in this case via the localhost.

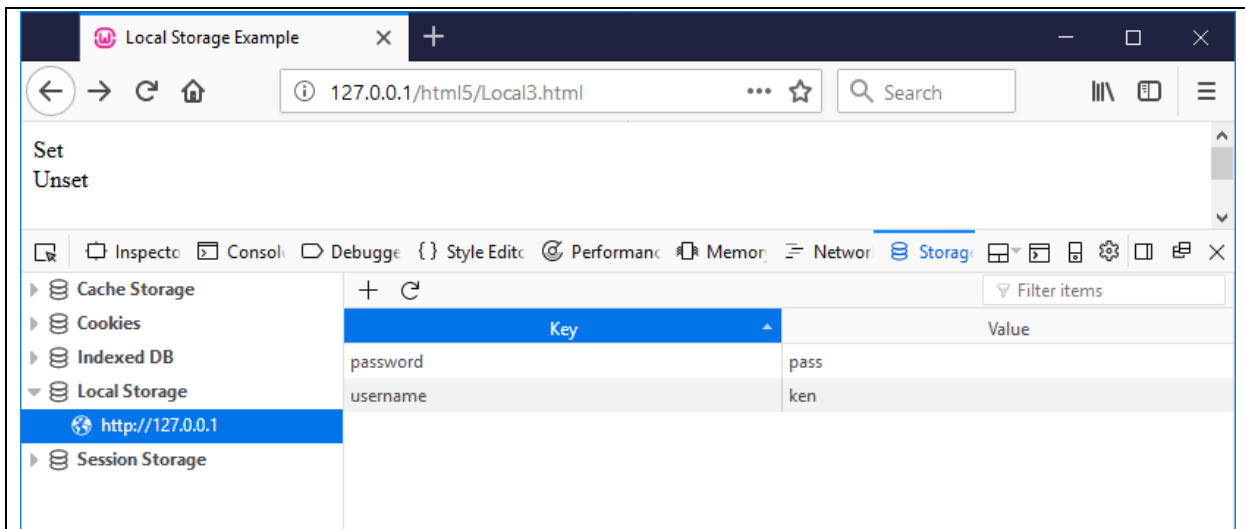


Figure 12.11: Local Storage

This example used Local Storage, which allows a lot more space to be available than traditional cookies, while cookies must be less than 4KB, Local Storage allows up to 5MB of data. The other alternative would be to use Session Storage, which as the name implies, means the data will exist for the duration of the session, until the browser window or tab is closed, while Local Storage persists indefinitely until the browser cache is cleared or the data is removed through JavaScript.

12.6 Web Workers

One of the problems with JavaScript involves concurrency as JavaScript is a single threaded environment, which means multiple scripts can't run simultaneously. Suppose the page needs to manipulate the DOM, handle UI events and process a large amount of data, unfortunately these tasks can't happen at the same time as one script may block another script from running. Fortunately HTML5 has added Web Workers, which allow scripts to be run in the background without hindering other scripts. Workers use message passing to send instructions to another thread for parsing, which is particularly useful when there are CPU intensive tasks needing to be run. In the following example a Web Worker is set up to calculate the factorial of a number. In reality, this is quite a trivial problem and a Web Worker wouldn't be necessary, so it is only intended as a demonstration of how Web Workers work, and how it could be applied to more complex problems such as retrieving and processing large amounts of data from the server, image processing, or mining Bitcoins.

In this example, the first step is to create a simple input form and a div to display the answer.

```
<input type="number" id="input">
<div id="answer"></div>
```

A Web Worker works by sending a message to a separate JavaScript file, in this case a file called 'worker.js'. This file will send responses when it is finished, without blocking JavaScript from continuing to run on the main page. The JavaScript begins by checking if the browser is compliant and capable of creating a worker. If so, a new Worker object is created with the specified JavaScript file. When the value in the input box changes a message is passed to the Worker using the `postMessage()` function. Finally an event listener is created to handle when the Worker sends its response. When the worker responds, the answer div is updated with the result.

```
if(window.Worker){
  var myWorker = new Worker('worker.js');
  $("#input").change(function() {
    myWorker.postMessage($("#input").val());
  });
  myWorker.addEventListener('message', function(e) {
    $("#answer").text(e.data);
  }, false);
}
```

The worker.js file for this example receives the value from the input box, and uses a simple recursive function to calculate the factorial, and returns the result using the `postMessage()` method.

```
onmessage = function(e) {
  var workerResult = 'Factorial: ' + fact(e.data);
  postMessage(workerResult);
}
function fact(input){
```

```
if(input<2){
  return 1;
}
else{
  return fact(input-1)*input;
}
>false);
}
```

The resulting page looks similar to Figure 12.12, but the calculation of the factorial is not being done on the page, and the result will be updated each time the Worker responds.



Web Workers can have a big impact on web application performance, and Web Workers can also invoke further Web Workers as well as making use of the XMLHttpRequest object. However, Web Workers are not able to access the DOM and make direct changes to a page, instead they should be considered a separate machine where work can be sent.

Key Points

- HTML5 is the latest HTML specification released in 2014 – a major update intended to catch up with the way the web has evolved.
- The new standard has some additional tags, including semantic tags designed to inform the machine what kind of data it is displaying, such as a header, footer, navigation, or the main section or article.
- Input types have improved to support different kinds of input.
- Rich media, such as audio and video, gains new support, removing the need for using browser plugins such as Flash.
- The canvas provides a space where graphics can be drawn, using code.
- Geolocation supports a mobile users allowing location based applications.
- Local Storage has transformed how cookies are used to store data on the client machine.
- Web Workers facilitate code to run in the background on the client browser.

Further Resources

- 1) W3C schools has an excellent introduction to the new features of HTML5. It can be found here:-
https://www.w3schools.com/html/html5_intro.asp
- 2) Web Fundamentals (formerly HTML5rocks) is a project run by google containing collection of tutorials, guides and best practices for futuristic webapps. It can be found here:-
<https://developers.google.com/web/>

- 3) Caniuse (Can I Use) is a useful site that shows the varying browser support for different features of HTML5. It can be found here:-
<https://caniuse.com/>

Assignment

Improve your calendar by allowing the user to choose different colours for different types of appointments. Allow users to share their calendar with friends and colleagues, and have their colleagues appointments appear in different colours. Senior users should be able to allow their secretaries to manage their appointments (create, edit etc.).

Chapter 13

Introducing Node.js

Objectives

This chapter begins introducing more modern web development techniques, and the MEAN (MongoDB, Express.js, Angular, Node.js) as an alternative to the traditional WAMP stack. Node.js is a free, open source, server-side environment offering an alternative to PHP, where JavaScript can be used throughout the stack. After reading this chapter you should:

- Understand the differences between Node.js and PHP.
- Be able to set up a Node.js environment.
- Parse URLs and handle basic routing using Node.js.
- Understand how server side events can be handled by Node.js.
- Be able to use the Node Package Manager (NPM) to add existing packages to a project.
- Send emails via Node.js.

Contents

13.1 PHP vs Node.js

13.2 Setting up Node.js

13.3 Events in Node.js

13.4 Node Package Manager (NPM)

13.5 Sending Emails with Node.js

So far this book has focused on the traditional server side technology stack of LAMP (or WAMP, or MAMP), meaning the operating system and then Apache, MySQL and PHP. This is a stable, reliable stack which powers a large portion of the web, with as much as 80% of websites running on PHP. This is perhaps due to the large number of applications built using PHP, including Content Management Systems (CMS) such as WordPress, Drupal and Joomla, with WordPress powering as much as 30% of the web. The most popular E-Commerce platforms such as WooCommerce, Magento and ZenCart are all written in PHP, and a variety of MVC (Model View Controller) Frameworks are popular using PHP, such as Laravel, Symfony, CodeIgniter, CakePHP and the Zend Framework. PHP will remain a very popular server side solution for the foreseeable future, however there is growing support for “JavaScript Everywhere”, with the capability of using JavaScript on the server as well, through the Node.js environment.

Node.js is a free, open source, server-side run-time environment for executing JavaScript on the server, allowing developers to use the same language on both the client and server.

Node.js is a free, open source, server-side run-time environment for executing JavaScript on the server, allowing developers to use the same language on both the client and server. One of the key advantages Node.js offers is handling requests without blocking. A common web server request is to retrieve a file and return the contents to the client machine. Using PHP (or ASP), this request generally takes a few steps;

- 1) Send the request to the server file system
- 2) Wait for the file system to open and read the file
- 3) Return the content to the client
- 4) Ready for another request

Node.js handles this crucially in a different way.

- 1) Send the request to the server file system
- 2) Ready to handle another request
- 3) When the file system has opened the file, the server sends the content to the client

Node.js eliminates the waiting between requests and is available to handle other requests. There are plenty of other differences that Node.js brings, making Node.js very important for modern web developers.

13.1 PHP vs Node.js

For a long time PHP and JavaScript have worked well together, with PHP managing the server and JavaScript managing the client, however since running JavaScript on the server became a possibility, the two languages have become alternatives for each other. PHP was created by Rasmus Lerdorf in 1994 and became the most popular server side language, powering many sites. Node.js is much more recent initially launched in 2009, but has rapidly gained popularity since then as an alternative choice to PHP. Both have advantages and

disadvantages, and as with many scenarios where developers have a choice between two languages, you will find staunch defenders of both and heated debate as to which is better.

The biggest selling point for Node.js is that it is asynchronous and non-blocking allowing it to service multiple concurrent events and various benchmark tests have demonstrated that Node.js is faster than PHP, even when using Facebook's HHVM. As server response times are vital, this is a major advantage. A second advantage is that using Node.js means only having one consistent programming language (JavaScript Everywhere), and data can be seamlessly moved between machines using JSON (JavaScript Object Notation), rather than developers needing to use different syntax in different places.

Node.js also boasts the NPM, a package manager, or large code repository, allowing developers to access a large ecosystem of open source libraries. While many of these packages are stable, core libraries, there are also many third party contributions, which are comparatively immature. Sometimes it can be difficult to assess the quality of the reusable code, and version inconsistencies along with complex code dependencies can lead to unexpected bugs and trickier code maintenance. On the other hand, as the code is newer it has been built with the latest architectural considerations in mind, such as Model View Controller (MVC). It is also better suited to state-of-the art features such as HTML5.

In comparison, PHP has a large, rich codebase with time tested code contributed by a large, experienced community. It was originally designed for being used on web servers and has evolved and updated over the years, working neatly alongside MySQL. There is a well-established hosting infrastructure in place, making it quick to get a project up and running, particularly when using a complete solution such as WordPress. Within PHP it is easy to switch between the content and the programming logic, by simply opening PHP tags when code is needed – whilst this is easy to do, unfortunately it leads to a poor separation of concerns making it harder to maintain.

Both PHP and Node.js have their supporters, and advantages and disadvantages, so both are likely to remain important components of web development for the foreseeable future.

13.2 Setting up Node.js

As with other programming challenges, the first step is to get “Hello World” to work, and make sure the environment is up and running. As with Wampserver, you can set up Node.js for testing on your local machine and have it run like the server – the first step is to download and install the latest version of Node.js from <https://nodejs.org>. Then create your first .js file as follows saving it as hello.js in a new directory on your machine.

```
var http = require('http');
http.createServer(function (req, res) {
  res.write('Hello World');
  res.end();
}).listen(8080);
```

The code instructs the server to output “Hello World” when a web browser accesses it using port 8080, but first it needs to be initiated by Node.js. To do this, open the command line interface, either through the

Node.js command prompt, or your system Command Prompt. Navigate to your new directory and type “node hello.js” to initiate Node.js.

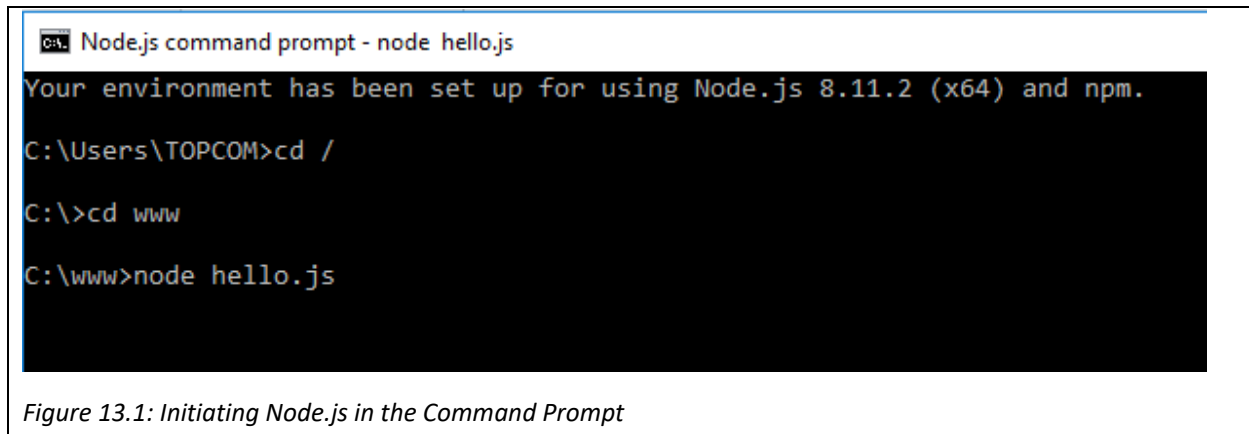


Figure 13.1: Initiating Node.js in the Command Prompt

Now when your computer is accessed via port 8080, Node.js will return a Hello World message. You can test this in your browser window by opening localhost:8080.

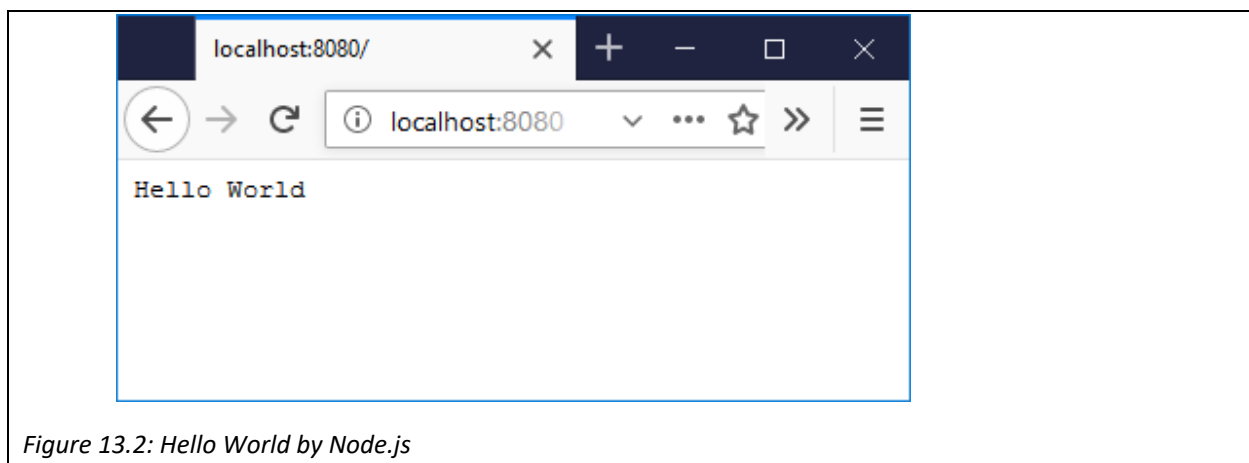


Figure 13.2: Hello World by Node.js

Let's examine the code in more detail to find out how it works. The first line requires the inclusion of a built-in Node.js module called 'http'.

```
var http = require('http');
```

Modules are much like JavaScript libraries, where useful functions can be stored. The 'http' module is one of the built-in modules which contains useful http functionality that can be used to build a webserver. As well as the built-in Node.js modules, you can create and require your own modules, or find modules that others have released to the npm. Having included the http module, a server can then be created.

```
http.createServer(function (req, res) {  
  res.write('Hello World');  
  res.end();  
}).listen(8080);
```

The `createServer()` method is called to create a HTTP Server object and turn the machine into a HTTP server. In this case the HTTP Server object is instructed to listen to port 8080. The object is passed as a parameter a `requestListener()` function which is executed whenever the server gets a request. This function contains two parameters, the request (`req`) and the response (`res`). The request parameter will contain things like the request method and the url. In this simplest of examples, the `write()` method is used to add the text "Hello World" to the response, and then the `end()` method is used to indicate that the response is complete and can be returned. The next example extends the previous example by first using the `writeHead()` method to first set the status code to be 200 (or OK), and add a response header to indicate the response is to be displayed as HTML. This time rather than outputting "Hello World", the rest of the URL is displayed instead.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

This will be displayed as in Figure 13.3, but clearly when the URL changes, the output will also change. Remember to initiate the new file using the command prompt.

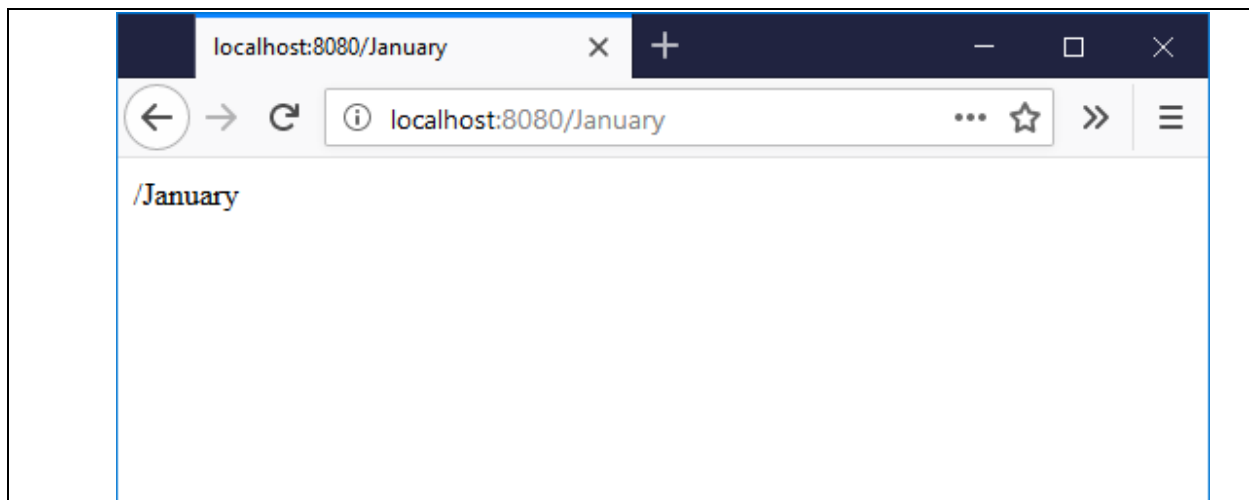


Figure 13.3: Extracting the URL in Node.js

Node.js also has a useful 'url' module, which can break up a url, extracting the host name, the pathname and any query parameters. After requiring the url module, the `parse()` method can be used to break it up, creating an object with the various parts of the url. The query part of the url can be returned either using `search`, or as a JavaScript object using `query`, as shown in the following example.

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var myURL = url.parse(req.url, true);
  res.write("Pathname: " + myURL.pathname + "<br>");
  res.write("Search: " + myURL.search + "<br>");
  var myURLData = myURL.query;
  res.write("Query: " + myURLData.month);
  res.end();
}).listen(8080);
```

Given the url <http://localhost:8080/index.html?month=January>, the following would be displayed.

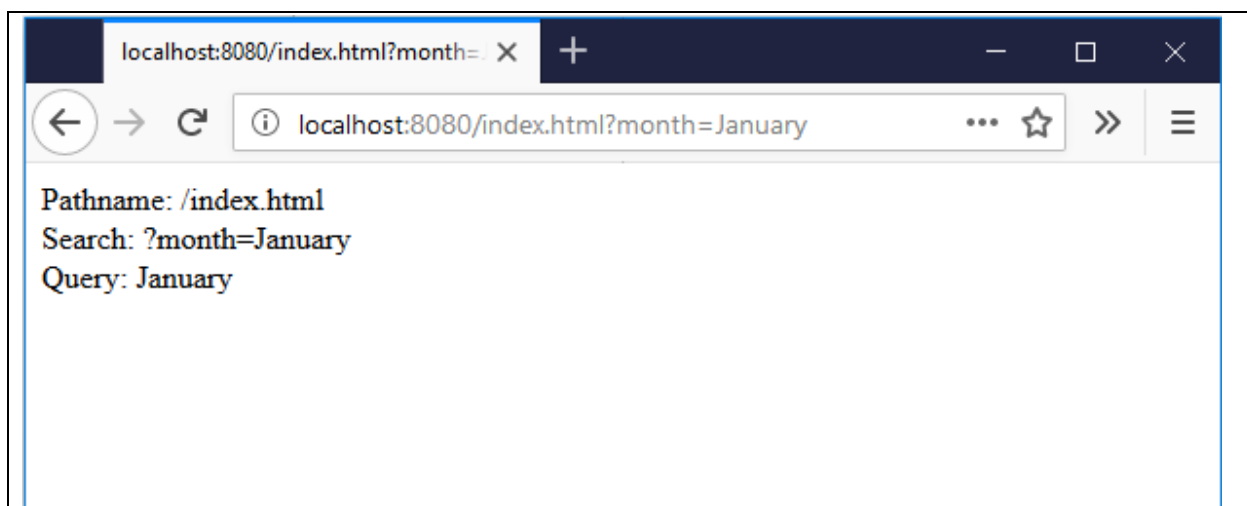


Figure 13.4: Parsing the URL in Node.js

Now that we can parse the URL, the next step is to set up the machine as a fileserver, that can return different pages based on the contents of the url. To do this, we require the file system module, or “fs”. The file system module can read, create, update, delete and rename files on the server, with the important part here being to read and return a file using the `readFile()` method. This this next example, the url is parsed to find the pathname so that a file can be specified – in this case the file should be in the same directory. The `readFile()` method is then used to try to read in data from the file. If there is an error, and the file can’t be found, the a 404 error is returned to the browser. On the other hand, if the file is located, its contents are sent back to the browser. In this way, Node.js is now set up as a web server, handling requests to the machine, locating the requested files and returning them, while managing errors if an incorrect url is requested.

```

var http = require('http');
var url = require('url');
var fs = require('fs');
http.createServer(function (req, res) {
  var myURL = url.parse(req.url, true);
  var file = "." + myURL.pathname;
  fs.readFile(file, function(err, data) {
    if(err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Nothing Found Here");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);

```

13.3 Events in Node.js

In JQuery we saw how JavaScript is used to handle browser events, such as a mouse click or keyboard press. Event Driven Programming is important whether in the browser or on the server, and Node.js has an events module to assist with managing events on the server. Many events can be fired on the server, such as when a file is opened for reading. Event Handlers can be created to managed events such as this, and an Event Emitter can be used to monitor events, while the Emit() method can be used to fire events. To use Node.js events, the built in 'events' module is required.

```

var events = require('events');
var eventEmitter = new events.EventEmitter();
var myEventHandler = function () {
  console.log('Do Something');
}
eventEmitter.on('Go', myEventHandler);
eventEmitter.emit('Go');

```

In this example an EventEmitter object is created. A handler (myEventHandler) function is written to log a simple statement in the console. The EventEmitter then associates the event handler with the 'Go' event. Finally the eventEmitter calls the emit() method to fire the 'Go' event. Figure 13.5 shows what happens when the code is initiated in the Node.js Command Prompt.

```

C:\www>node event.js
Do Something

```

Figure 13.5: A Console Log Event in Node.js

To give a clearer example of how this event handling might be useful, consider a group chat room where users can join and leave. The chats will be handled by the server, but it would be useful to alert other group members when users join or leave the room. This example assumes that an `alertUsers()` function has been written, and is triggered by either the `userJoin` or `userLeft` event.

```
var events = require('events');
var chatEvents = new events.EventEmitter;
function userJoin(username){
  alertUsers('User ' + username + ' has entered.');
```

```
}
function userLeft(username){
  alertUsers('User ' + username + ' has left.');
```

```
}
chatEvents.on('userJoin', userJoin);
chatEvents.on('userLeft', userLeft);
```

The remaining step is to create a trigger for the `userJoin` and `userLeft` events, by adding the `emit` method call to the login and logout functions, as follows.

```
function login(username){
  chatEvents.emit('userJoin', username);
}
function logout(username){
  chatEvents.emit('userLeft', username);
}
```

The examples above have demonstrated that the Node.js `EventEmitter` has several methods, demonstrating the key `on()` and `emit()` methods. There are also methods to remove listeners from reacting to events.

13.4 Node Package Manager (NPM)

The NPM is a package manager for Node.JS packages or modules. As well as the core modules, some of which we have discussed already, there are thousands of contributed modules which can be easily incorporated into a project. When you installed Node.JS you automatically got access to the package manager, which makes it easy to include more modules in your website. You can search for modules to include at www.npmjs.com. Suppose we wanted to convert numbers into fractions – one option would be to write our own module to solve this problem, but a better option would be to search the NPM to reuse someone else’s existing solution.



Figure 13.6: Sample Module from NPM

Figure 13.5 shows a screen shot of a possible module included in the NPM. As well as giving a brief description, it also gives some guidance about the quality of the module. In this case the bars on the right of the screen suggest the popularity is 45%, the quality is 97% and the maintenance is 100%. Selecting the module provides more information about how to use it, and information about any dependencies or dependents the module may have. In the case of num2fraction, there are no Dependencies, so we don't need any further modules to make it work. There are however 17 Dependents, modules which rely on num2fraction to work.

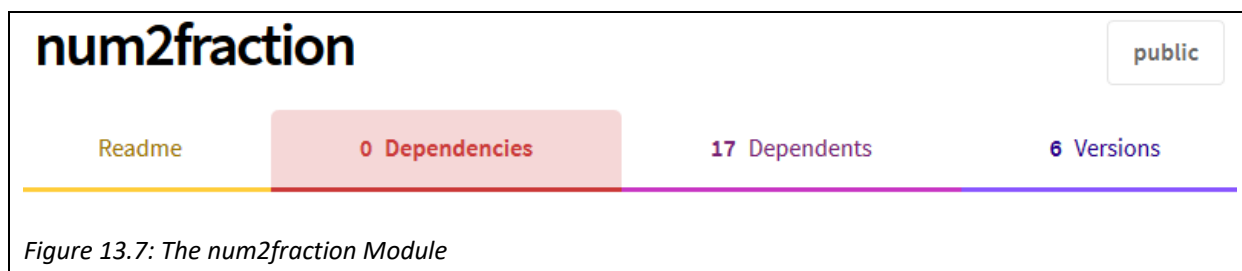


Figure 13.7: The num2fraction Module

Having selected a module, the next step is to use npm to install it, using the Node.js command prompt. For this module type:-

```
C:\www>npm install num2fraction
```

At this stage you may get some warning messages, but wait for a few moments for the module to download and set up and then you should notice some additional parts to your working directory. Firstly a new directory is created called node_modules, and this is where all future modules will be installed. If you open this directory, you'll notice the num2fraction module is already there in its own directory. A package-lock.json file has also been added to the main working directory. This is there to help maintain consistency as your project grows. For now we want to test the num2fraction module.

```
var http = require('http');
var n2f = require('num2fraction');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(n2f(0.5));
  res.end();
}).listen(8080);
```

In this code we require the num2fraction module, and then write `n2f(0.5)` to the screen. As expected the following should appear.

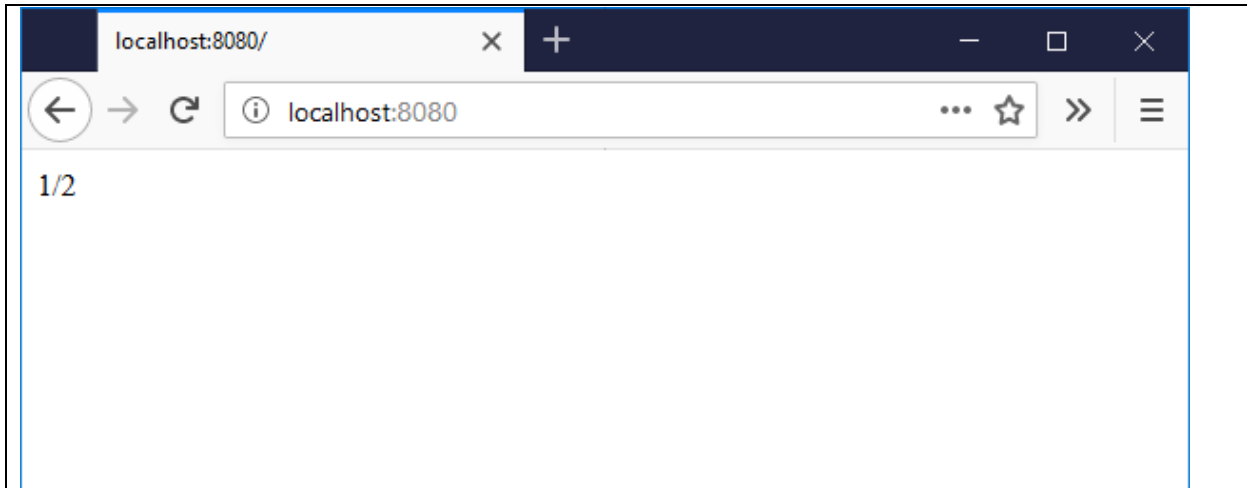


Figure 13.8: Demonstrating the num2fraction Module

13.5 Sending Emails with Node.js

Node.js also makes it fairly easy to set your machine up as an email server. The first step is to install the nodemailer module via the npm using the following command.

```
C:\www>npm install nodemailer
```

The following code is a fairly standard example of how to send an email from a gmail account. Clearly the username and password would need to be edited, along with the subject, text and to fields. Beyond that, the example is fairly straightforward.

```
var http = require('http');
var nodemailer = require('nodemailer');
var transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'myemail@gmail.com',
    pass: 'secretpass'
  }
});
var mailOptions = {
  from: 'myemail@gmail.com',
  to: 'myfriend@yahoo.com',
  subject: 'Welcome to the web',
  text: 'Feel free to explore!'
};
```

```
transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
  } else {
    console.log('Email sent: ' + info.response);
  }
});
```

Key Points

- Node.js is a popular, modern server-side runtime environment written in JavaScript, which is fast due to its ability to handle requests without blocking.
- Node.js can be used to handle Http requests, by creating a server and parsing the requested URL.
- Node.js can be set up as a file server to return different files depending on the Http request.
- The emit() method can be used to respond to server-side events, such as when a user enters or leaves a chat room.
- The Node Package Manager (or NPM) is a large collection of modules of code that are publicly available and easily installed using npm.

Further Resources

- 1) Node.js can be downloaded from the following link, which also has complete documentation.
<https://nodejs.org/en/>
- 2) Lots of information about available modules in the npm can be found here.
<https://docs.npmjs.com/>
- 3) W3schools offers an introductory tutorial to Node.js here.
https://www.w3schools.com/nodejs/nodejs_intro.asp

Assignment

Set up a simple Node.js environment where a user is directed to different pages based on the parameters in their url. Make sure you are set up to use the Node Package Manager as it will be necessary in subsequent chapters.

Chapter 14

Node.js, MySQL & MongoDB

Objectives

This chapter continues discussing the use of Node.js as the server-side scripting language. While the previous chapter introduced using Node.js to manage the server's filesystem, a database is normally a crucial part of most web applications. This chapter introduces how Node.js can work alongside a MySQL database, and also introduces MongoDB, a NoSQL database that is growing in popularity. The uses of both alternatives are discussed, as well as a comparison between them. After reading this chapter you should:

- Be able to perform basic operations on a MySQL database using Node.js (create, insert, retrieve).
- Understand the differences between MySQL and MongoDB.
- Be able to perform basic operations on a MongoDB database using Node.js.

Contents

14.1 Node.js & MySQL

14.2 Node.js & MongoDB

14.3 MySQL vs MongoDB

With Node.js powering the server, as well as managing the file system, it also needs to work alongside a database. Previously, in chapter 6, we explored using PHP alongside a MySQL. This chapter demonstrates first using Node.js as the scripting language for accessing a MySQL database, and then later using MongoDB as a document store instead. A comparison of both database platforms is also discussed.

14.1 Node.js & MySQL

MySQL is a popular open source relational database management system, which has become a reliable choice since its initial release in 1995. Today it remains a core component of the LAMP stack (as discussed previously), as well as being the database used in Wordpress, Drupal, Joomla and many other key web applications. As seen in chapter 6, MySQL stores data in tables of similar types of data, with perhaps a table for users and another table for appointments. The user table would then store a collection of records about each user, with a record comprising of several fields, such as id, username, password etc. MySQL works the same way with Node.js, we just use Node.js to access it.

14.1.1 Connecting to MySQL

The first step is to download the latest version of MySQL from <https://www.mysql.com/downloads/>. Once you've installed it on your machine, you can access it via Node.js. To do so, a mysql driver is required, which can be done by downloading and installing the mysql module using the NPM and the command terminal.

```
C:\www>npm install mysql
```

Once installed, Node requires this module in its file to allow access to further functions, such as the `createConnection()` function.

```
const mysql = require('mysql');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'xxxxxxx'
});
connection.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  connection.query("CREATE DATABASE calendar", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

This example creates a connection to the database to make sure it is set up correctly. If there is a problem with the setup, an error will be displayed in the command prompt to help you fix the problem. The `createConnection()` function sets up the host, username and password for the connection, which obviously need to be edited. Here a database can also be specified, but in this example no database has been created yet.

In this example, once the connection is established using the connect() function, a query is executed to create a database called calendar. Running this script in the command prompt should result in two successful console logs.

```
C:\www>node db.js
Connected!
Database created
```

Previously we introduced the phpmyadmin GUI tool which provides support for the database. Clearly if we are using Node.js rather than PHP, it isn't worth installing the LAMP stack in order to check the database content. For these examples we shall continue accessing the database via code, however if a GUI visualization is preferred, the MySQL Workbench comes as part of the MySQL installation, and after running the previous code, you should see the calendar database schema appear in the workbench, as shown in Figure 14.1.

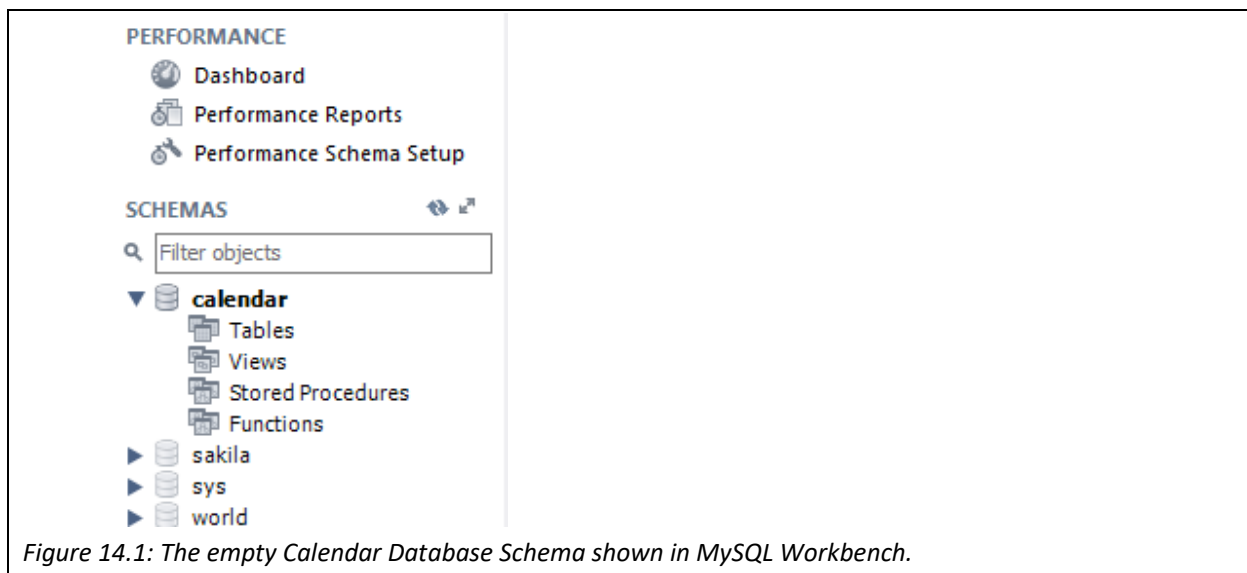


Figure 14.1: The empty Calendar Database Schema shown in MySQL Workbench.

14.1.2 Creating a MySQL Table

Once the database has been created the next step is to add a table to it. Again, Node.js is used to create a table. Note the difference in the createConnection() function, with the database being specified.

```
const mysql = require('mysql');
const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "xxxxxxx",
  database: "calendar"
});
```

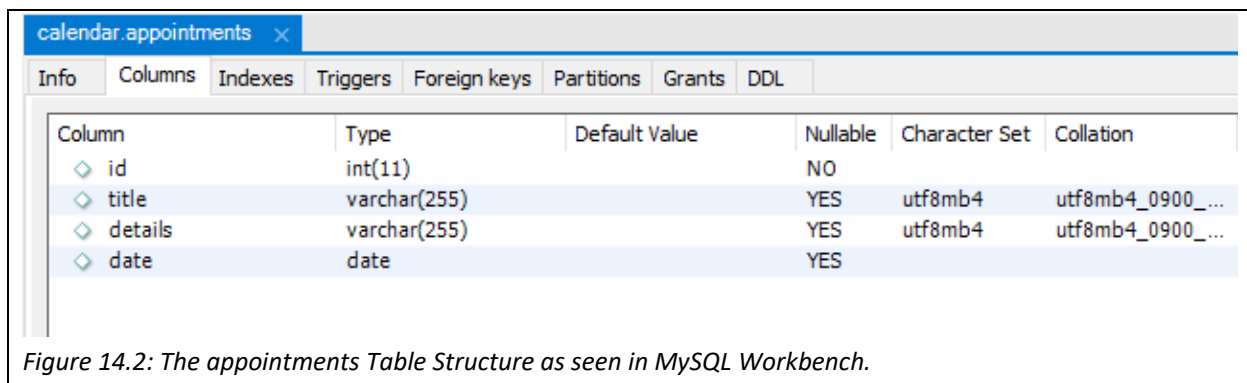
The SQL statement for creating a table has been discussed previously, in this example a SQL query is constructed and then called by the query() method.

```

connection.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE appointments (id INT AUTO_INCREMENT PRIMARY KEY, title VARCHAR(255), details VARCHAR(255), date DATE)";
  connection.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});

```

Running the file in the Node.js command prompt, should result in the successful console logs being displayed. In this case, the table has 4 fields, beginning with an ID field which is automatically incremented. The other fields are for storing the title, details and date of an appointment. Again, the MySQL Workbench can be used to check the table is set up as planned.



The screenshot shows the MySQL Workbench interface for the 'calendar.appointments' table. The 'Columns' tab is selected, displaying a table with the following structure:

Column	Type	Default Value	Nullable	Character Set	Collation
id	int(11)		NO		
title	varchar(255)		YES	utf8mb4	utf8mb4_0900_...
details	varchar(255)		YES	utf8mb4	utf8mb4_0900_...
date	date		YES		

Figure 14.2: The appointments Table Structure as seen in MySQL Workbench.

14.1.3 Inserting Data into MySQL

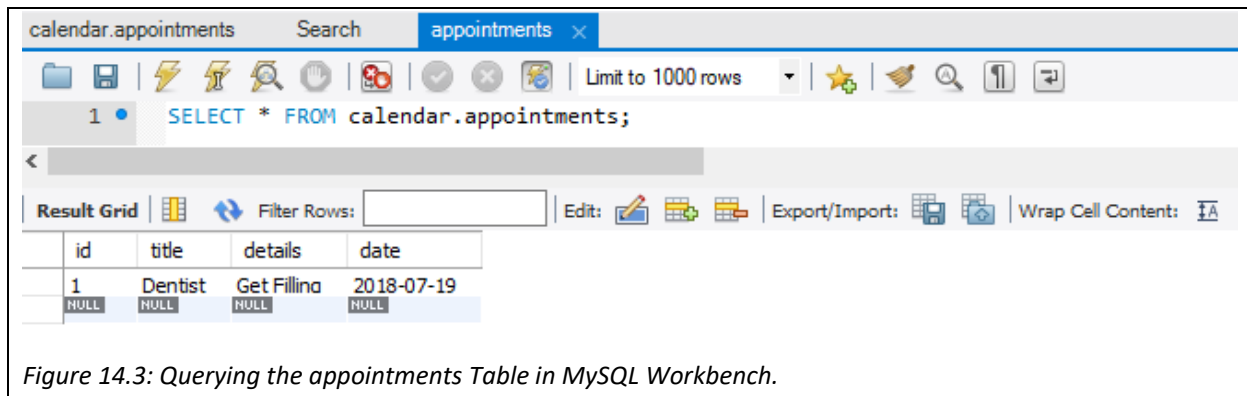
Having created the table, the next step is to insert some data.

```

connection.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO appointments (title, details, date) VALUES ('Dentist', 'Get Filling', '2018-07-19 03:00:00')";
  connection.query(sql, function (err, result) {
    if (err) throw err;
    console.log("1 record inserted");
  });
});

```

Again, the SQL statement is prepared and then sent to the database via the query() method. Again the Node.js console will output a success message and the MySQL Workbench can be used to reconfirm success.



14.1.4 Retrieving Data from MySQL

We can retrieve the data from the database using the standard SQL Select statement.

```
connection.connect(function(err) {
  if (err) throw err;
  connection.query("SELECT title, details, date FROM appointments", function (err, result, fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

Running this code from the Node.js Command Prompt presents the current database contents.

```
C:\www>node db.js
[ RowDataPacket {
  title: 'Dentist',
  details: 'Get Filling',
  date: 2018-07-18T17:00:00.000Z } ]
```

Notice the callback function for the query takes 3 parameters; err, result and fields. The err parameter contains any errors in case that the query fails, and hence in the example the error is thrown to the console. The result parameter is displayed above and in this example there is only one entry in the database to be returned. The result object is an array where each row is stored in a different index, so the contents of the array could be written to a webpage as follows, with the result displayed in Figure 14.4.

```
res.write(result[0].title + "<br>" + result[0].details + "<br>" + result[0].date);
```

localhost:8080

Dentist
Get Filling
Thu Jul 19 2018 00:00:00 GMT+0700 (SE Asia Standard Time)

Figure 14.4: Outputting the Results of a Query.

The third parameter, fields provides some information about each field that is returned, so for example if that is logged for our example query, the fields data about the name field is as follows.

```
catalog: 'def',
db: 'calendar',
table: 'appointments',
orgTable: 'appointments',
name: 'title',
orgName: 'title',
charsetNr: 33,
length: 765,
type: 253,
flags: 0,
decimals: 0,
default: undefined,
zeroFill: false,
protocol41: true },
```

The query function can support any valid SQL statements, including Update, Delete, Drop Table, and clauses such as Limit and Where, as well as handling table Joins. As an example, we can edit the SELECT statement to include a where clause to select appointments from a particular date.

```
connection.query("SELECT * FROM appointments WHERE date='2018-07-19'", function (err, result, fields) {
  if (err) throw err;
  console.log(result);
});
```

The threat of SQL injections has already been discussed, but it is a common problem to query a database based on a user input. It is important to escape any user inputted values to prevent an SQL injection. This can be done using the mysql escape method, as in the following example where the query is based on a variable 'date'.

```
connection.query("SELECT * FROM appointments WHERE date=" + mysql.escape(date), function (err, result,
fields) {
  if (err) throw err;
  console.log(result);
});
```

14.2 Node.js & MongoDB

Just as Node.js offers an alternative to PHP as a server side language, there are alternative databases to MySQL, with MongoDB becoming the leading NoSQL database. Rather than using the traditional relational data model, NoSQL databases offer more flexibility to how data is stored and retrieved, helping to assist with the modern world challenges of 'Big Data'. MongoDB therefore became one of the key elements in the MEAN development stack, which consists of MongoDB, Express.js, Angular and Node.js. The MEAN stack is growing in popularity for developing modern web apps and the other parts of the stack are discussed in following chapters.

14.2.1 Connecting to MongoDB

There are several platforms setup to run a MongoDB database as a service in the cloud, via Amazon or Google. Alternatively MongoDB can be downloaded from <https://www.mongodb.com>, with different versions available for different platforms. Follow the installation instructions to setup MongoDB on your local machine to experiment with the examples discussed here. MongoDB requires a directory to store its data in – in this example, it was set to `\data\db`. To start MongoDB on a windows machine, the `mongod.exe` executable needs to be run. The database can then be accessed through another command prompt by opening `mongo.exe`. As well as installing MongoDB, the `mongoose` module needs to be installed by Node.js so that we can access it via Node.js scripts.

```
C:\www>npm install mongoose
```

Once set up, the following script will connect to the database.

```
var MongoClient = require('mongoose').MongoClient;
MongoClient.connect("mongoose://localhost:27017/myDB", function(err, db) {
  if(!err) {
    console.log("Connection Successful");
  }
});
```

14.2.2 Creating a Collection in MongoDB

MongoDB stores its data in collections, which are similar to tables in traditional relational databases. Each database can have several collections, and each collection contains data in documents, similar to records in a relational database. Rather than needing create the structure of a table, data of a variety of types can simply be added to a collection. The following example then creates a database called 'calendar', and then creates a collection called 'appointments'. If an error occurs a message will be posted in the Node console.

```
var MongoClient = require('mongoose').MongoClient;
MongoClient.connect("mongoose://localhost:27017/calendar", function(err, db) {
  if (err) throw err;
  const myDB = db.db('calendar');
  myDB.createCollection('appointments', {strict:true}, function(err, collection) {
    if (err) throw err;
  });
});
```

The method called `createCollection()` is used here to create the collection, and in this case it takes 3 parameters, the first being the name of the new collection, the second `{strict:true}` will make sure the collection doesn't already exist and returns an error if it does. The final parameter is a call back, which in this case is just set to display an error. Once the collection is created it can be connected to by a shorter connection method call.

```
myDB.collection('appointments', {strict:true}, function(err, collection) { });
```

14.2.3 Inserting Data into MongoDB

Once the collection is created data can be stored in it, by using the insert method. Note that it isn't necessary to specify what type of data is being inserted, or setup the collection structure before inserting data. This makes MongoDB particularly useful for managing unstructured and semi-structured data.

```
var MongoClient = require('mongodb').MongoClient;
MongoClient.connect("mongodb://localhost:27017/calendar", function(err, db) {
  var myDB = db.db('calendar');
  var collection = myDB.collection('appointments');
  var doc1 = { date: "2018-06-28", title: "Dentist", details: "Get Filling Done" };
  collection.insert(doc1);
});
```

MongoDB Compass is a convenient way to manage the data in a MongoDB database, similar in some ways to the phpmyadmin tool introduced previously. Compass offers a GUI that allows you to visualize and explore the data, as well as inserting, deleting and updating records. For now, it is useful for checking that the database and collection have been created correctly and the data has been inserted. Figure 14.5 shows a screen shot of compass with the data correctly inserted - note that the document has been given a unique identifier (5b50a81472139025b80c58f2 in this example).

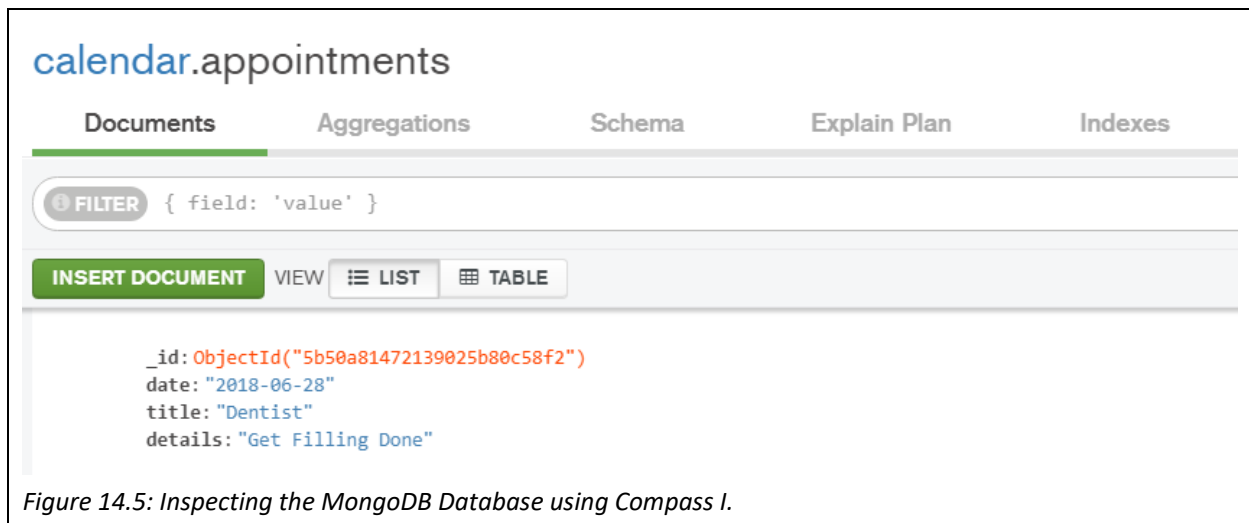


Figure 14.5: Inspecting the MongoDB Database using Compass I.

MongoDB inserts (and updates and removes) data in the database asynchronously, which means the command is sent to the server and then forgotten, which means there is a possibility that the data doesn't get inserted for some reason. The insert() method can be enhanced by adding a callback function, and also adding {w:1} to ensure an error is returned if the document fails to insert correctly. In this and subsequent examples the requires to connect to the database are omitted.

```

var collection = myDB.collection('appointments');
var doc1 = { date: "2018-06-30", title: "Birthday" };
collection.insert(doc1, {w:1}, function(err, result) {
  if(err) throw err;
  else
    console.log("Success!")
});

```

Also notice that in this second example there is no 'details' field. MongoDB also allows multiple documents to be inserted in one statement by creating an array of documents.

```

var collection = myDB.collection('appointments');
var docs = [{ date: "2018-07-01", title: "Meeting", details: "In Room 412" },
  { date: "2018-07-02", title: "Follow Up", details: "In Room 412" } ]
collection.insert(docs, {w:1}, function(err, result) {
  if(err) throw err;
  else
    console.log("Success!")
});

```

Again Compass can be used to inspect the inserted data.

	_id ObjectId	date String	title String	details String
1	5b50a81472139025b80c58f2	"2018-06-28"	"Dentist"	"Get Filling Done"
2	5b50a881b8e643198c2e7ec0	"2018-06-30"	"Birthday"	No field
3	5b50a8aebf1592c247d700a	"2018-07-01"	"Meeting"	"In Room 412"
4	5b50a8aebf1592c247d700b	"2018-07-02"	"Follow Up"	"In Room 412"

Figure 14.6: Inspecting the MongoDB Database using Compass II.

14.2.4 Retrieving Data from MongoDB

The simplest way to retrieve data from a MongoDB Database is using the find() method.

```

var collection = myDB.collection('appointments');
collection.find().toArray(function(err, items) {
  console.log(items);
});

```

In this example the entire contents of the collection is returned and converted into an array, which could then be processed perhaps to search for appointments from a particular day. Care should be taken with this approach however as the memory requirements for handling large amounts of data may create severe scalability concerns. Nonetheless the console can be used to inspect the contents of the database as follows.

```
[ { _id: 5b50a81472139025b80c58f2,
  date: '2018-06-28',
  title: 'Dentist',
  details: 'Get Filling Done' },
  { _id: 5b50a881b8e643198c2e7ec0,
  date: '2018-06-30',
  title: 'Birthday' },
  { _id: 5b50a8aeebf1592c247d700a,
  date: '2018-07-01',
  title: 'Meeting',
  details: 'In Room 412' },
  { _id: 5b50a8aeebf1592c247d700b,
  date: '2018-07-02',
  title: 'Follow Up',
  details: 'In Room 412' },
```

An alternative is to use the `findOne()` method where a query field can be specified and only one result returned.

```
var collection = myDB.collection('appointments');
collection.findOne({date:'2018-06-28'}, function(err, item) {
  console.log(item);
});
```

In this case just a result matching the date is returned.

```
{ _id: 5b50a81472139025b80c58f2,
  date: '2018-06-28',
  title: 'Dentist',
  details: 'Get Filling Done' }
```

The preferred method of querying a MongoDB database is to stream results in and deal with them individually, rather than selecting all the data in one go.

```
var collection = myDB.collection('appointments');
var stream = collection.find({date: /. *2018\06.* /}).stream();
stream.on("data", function(item) {
  console.log(item);
});
stream.on("end", function() {
  console.log("Done");
});
```

A stream is created to collect and process the results. In this case a regular expression is used to find any results where the date contains 6/2018. When the stream returns data it is logged in the console, and when it ends the console logs "Done".

```
{ _id: 5b50a81472139025b80c58f2,
  date: '2018-06-28',
  title: 'Dentist',
  details: 'Get Filling Done' }
{ _id: 5b50a881b8e643198c2e7ec0,
  date: '2018-06-30',
  title: 'Birthday' }
Done
```

The query parameter specified in the find() method allow a subset of results to be returned, and MongoDB supports several comparison operators as shown in Table 14.1.

Operator	Description
\$eq	Matches values equal to specified value
\$gt	Matches values greater than specified value
\$gte	Matches values greater than or equal to specified value
\$in	Matches any values in a specified array
\$lt	Matches values less than specified value
\$lte	Matches values less than or equal to specified value
\$ne	Matches values not equal to specified value
\$nin	Matches values not in a specified array

Table 14.1 MongoDB Query Comparison Operators

14.2.5 Deleting Data from MongoDB

Records can be removed from MongoDB using the remove() method.

```
var collection = myDB.collection('appointments');
collection.remove({date:'2018-06-28'}, {w:1}, function(err, result) {
  if(err) throw err;
  else
    console.log("Success!")
});
```

Note that this example uses a callback to ensure that the data is removed. Also note that if no parameters are given to the remove() method, all the documents in the collection will be removed.

14.2.6 Updating Data in MongoDB

Data can be updated in a MongoDB database using the update method(), as demonstrated in the following trivial case.

```
var collection = myDB.collection('appointments');
collection.update({date:'2018-06-30'}, {$set:{date:'2018-06-29'}}, {w:1}, function(err, result) {
  if(err) throw err;
  else
    console.log("Success!")
});
```

14.3 MySQL vs MongoDB

So far this chapter has illustrated how Node.js can be used alongside both a MySQL and a MongoDB database, and while either would work as the database, there are clearly similarities and differences between them (and other alternatives). Some obvious similarities are that both are inexpensive database solutions, and both support basic CRUD operations (Create, Read, Update, Delete) on the data. There are also some clear differences, in terms of both syntax and semantics. While MySQL deals with fields and tables, MongoDB deals with documents and collections. Syntactical differences include the difference between “SELECT” and “find()”. Regardless, both provide a valid database solution, so it is worth exploring the differences between them and their relevant use cases.

Relational databases have been the stalwart of databases for many decades, with MySQL being an established choice since 1995. Relational databases are carefully designed to structure and manage data in a clear, organized way. Over recent years, data has changed in several ways. Firstly the concept of “Big Data” has emerged, describing an environment where much more data is being produced concerning a wide variety of topics. In today’s interconnected world, every interaction creates data of some kind, and there is a need for managing and organizing such data to extract valuable information. Databases today are being expected to deal with different kinds of data; unstructured and semi-structured. MySQL and other relational databases have great strengths to offer, but NoSQL databases such as MongoDB offer alternative benefits.

One advantage MongoDB has over MySQL is the dynamic schema which affords flexibility in the way the database is designed. The flexibility supports missing data and also faster agile development methods as there is no need to design and define the database schema before development. Modern application development is fast-paced with the need to deliver a product to market being a pressing issue, and MongoDB supports agile development. The data in MongoDB is stored as BSON, a variation of JSON that supports all JavaScript data types.

On the other hand, MySQL is a mature solution that is well tested, and built to be compatible with many environments. MySQL follows the ACID model, which means consistency in the database is Atomic, Consistent, Isolated and Durable, while MongoDB currently follows a BASE model, which means Basic Availability, Soft-state and Eventual consistency. Because of this MySQL is more likely to yield accurate results, while it is acceptable for some updates to not be immediate in MongoDB. However, MongoDB is scheduled to follow the ACID model in 2018.

A key difference between the databases concerns scalability, which is particularly an issue in the current world of “Big Data”. MySQL relies on “Vertical Scaling”, where the database is stored on a machine and the performance of that machine is improved by increasing the RAM and processing capabilities of that machine. “Horizontal Scaling”, which is better supported by MongoDB, involved dividing the data between multiple machines where each machine handles their share of the workload, otherwise known as sharding. MongoDB’s ability to break up a dataset and store it in multiple shards supports scalability.

Selecting the right database for a new project is an important decision, and both relational databases like MySQL and NoSQL databases like MongoDB offer valid alternatives. Legacy projects with well-established

structure for their data will certainly prefer a relational data model, while many new projects dealing with unstructured data, with an expectation to grow fast may choose a database such as MongoDB.

Key Points

- MySQL is a relational database where similar types of data is stored in tables that are then related to each other.
- MySQL Workbench can be used to view a MySQL database.
- Standard MySQL queries can be executed by Node.js to create, insert and retrieve data from a MySQL table.
- MongoDB offers a flexible data store where data is stored in collections, adapting to challenges of Big Data.
- The MongoDB Compass tool can be used to view a MongoDB database.
- MongoDB queries are similar to MySQL queries, allowing the same abilities to create, insert, retrieve and delete data.
- MySQL and MongoDB both have their uses in modern web development, understanding their strengths is important when deciding which option to use.

Further Resources

- 1) W3Schools has an introduction to using Node.js with MySQL that can be found here:
https://www.w3schools.com/nodejs/nodejs_mysql.asp
- 2) MongoDB can be downloaded from here, along with further relevant resources:
<https://www.mongodb.com/>
- 3) Tutorialspoint offers a comprehensive introduction to MongoDB here:
<https://www.tutorialspoint.com/mongodb/>

Assignment

Previous chapters should mean you are already comfortable with MySQL as a database, so set up a MongoDB collection of calendar appointments and adjust your existing code to populate your calendar with appointments.

Chapter 15

Express.js

Objectives

Express.js is a lightweight framework for Node.js. A previous chapter introducing Node.js introduced some of the available modules from the Node Package Manager and discussed some server tasks such as handling URL requests. This chapter extends on that as using a framework makes tasks such as routing more convenient. Express.js is part of the MEAN development stack, along with MongoDB, Node.js and Angular. This chapter will discuss how Express.js works alongside both Node.js and MongoDB to handle server requests. After reading it you should:-

- Be able to set up routes for handling different HTTP methods.
- Be able to build an API to make database requests easier to manage by the client.
- Be able to add further modules to Express.js, such as body-parser, which can parse the contents of a form to be inserting into a database.
- Be able to handle files uploaded to the server by the client.

Contents

15.1 Getting Started with Express.js

15.2 Routing with Express.js

15.3 Creating an API with Express.js

15.4 Uploading Files with Express.js

Express.js is described as a fast, unopinionated, minimalist web framework for Node.js. This description may need some further explanation. Firstly, Express.js is billed as a framework, while jQuery is billed as a library and Node.js is billed as a runtime environment. There is some debate over the use of these different terms, most notably the difference between a library and a framework. The generally accepted difference between a library and a framework goes along the lines of “don’t call us, we’ll call you”, in that a library contains a collection of code that can be called by the developer, while a framework provides generic functionality that can be built upon by the developer. Frameworks have some differences from libraries, in that there is an inversion of control – while the developer calls the library, the framework ‘calls’ the developer’s code. Frameworks are also intended to be extensible, but not modifiable, in other words, the developer can add to the existing code, but not change the existing code.

Express.js is a fast, unopinionated, minimalist web framework for Node.js

Express.js is also fast, and minimalistic, and intentionally so. There are various further modules that can be added to extend the core functionality, but Express.js is unopinionated, leaving it up to the developer to choose which modules to use and how to implement them. Express.js therefore gains the benefit of implementing good practice, and good software design patterns, while allowing the developer some flexibility. This chapter will investigate how Express.js builds on Node.js.

15.1 Getting Started with Express.js

The first project as always is to display “Hello World”, this time using Express.js. First create a new directory for your project, then open your Node.js console and navigate to the right directory. This time we will use Node.js to create the package.json file, using the command “npm init”. The package.json file is particularly useful when you want to publish a project, so you will be asked to give your package a name, version, description, etc. For now it is ok to just use the default settings, although you should name your package something other than express – perhaps expressdemo. The next step is to install express to your project.

```
C:\www\expressdemo>npm install express
```

The next step is to create a server.js file in the root of your new directory, with the contents as follows.

```
const express = require('express');
app = express();
app.get('/', function(request,response){
  response.send("Hello World!");
});
app.listen(3000, function(){console.log("Express server started");});
```

There is similarity between this example and the “Hello World” example in Node.js, in this case though Express.js is used to send the response. Notice that the code does require the “http” module this time, this is

because Express.js already requires it. This code responds to a get request on port 3000. We can initialize it in the Node.js terminal to get the console message declaring the server has started.

```
C:\www\expressdemo>node server.js
Express server started
```

Opening localhost in a browser using port 3000 will then display “Hello World!”.

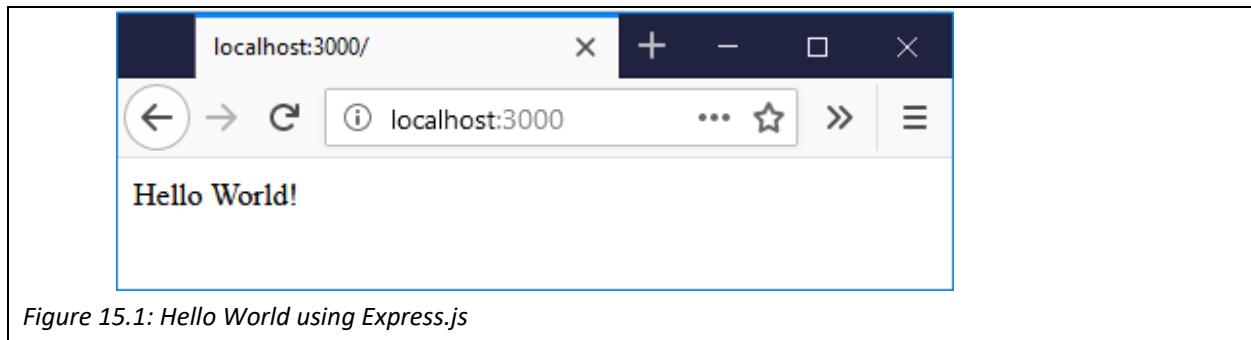


Figure 15.1: Hello World using Express.js

15.2 Routing with Express.js

Routing is concerned with how the application endpoint handles requests from the client and the server needs to be instructed how to deal with requests correctly. In the preceding example this was managed by the `get()` method, which handles GET requests. Here when the root of the domain is called (localhost), a callback function is specified to send “Hello World!” back as the response. The Express.js app object contains methods for handling each of the HTTP methods, GET, POST, etc. and the basic syntax for each method is as follows;

```
app.METHOD('path', callback);
```

Multiple callbacks can be specified in sequence for each method and if so the `next()` function should be called to move on to the next callback function. In the above example the callback function took the request and response objects as parameters. The request object contains all the information about the request, including any GET parameters sent, for example, the request object contains a `params` variable that is demonstrated in the next example.

```
const express = require('express');
app = express();
app.get('/:month/:year', function(request, response){
  response.send(request.params);
});
app.listen(3000,function(){console.log('Express server started')});
```

Once again initialize this new version of `server.js` in the Node.js console and then opening up a new browser will display the request parameters. This time the url isn't the root url, instead it contains a path specifying the month and year such as:- `http://localhost:3000/12/2018`. These parameters are then sent back

in the response and displayed as JSON in the browser window as in Figure 15.2. The next step will be to use the contents of the request object to specify different routes.

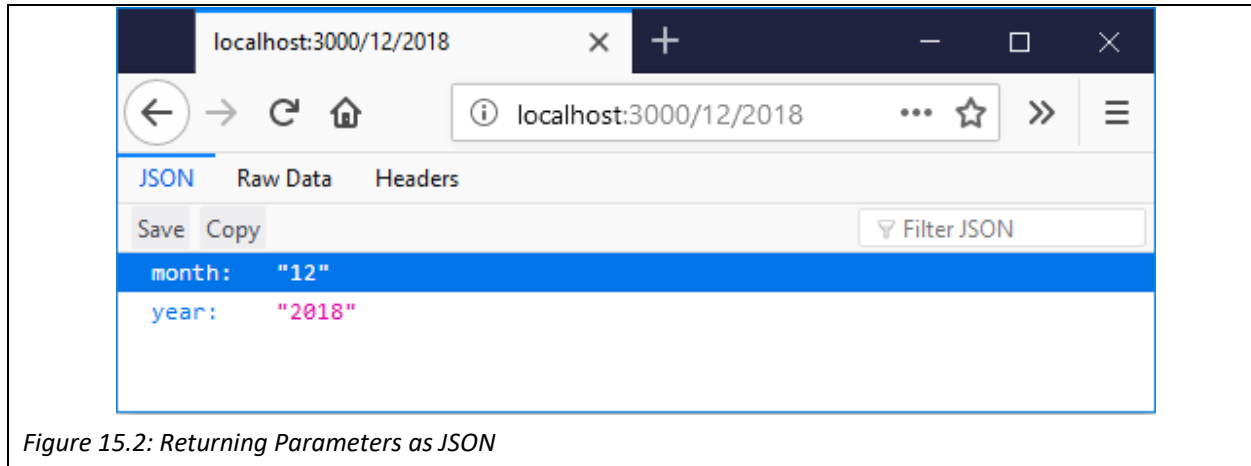


Figure 15.2: Returning Parameters as JSON

Once again, editing the server.js file we could specify different routes for different URLs to take. In this case the route resolves with a simple output, but depending on the url, different outputs will be displayed.

```
const express = require('express');
app = express();
app.get('/login', function(request, response){
  response.send("Login Page");
});
app.get('/users/ken', function(request, response){
  response.send("Ken's Home Page");
});
app.get('/users/matty', function(request, response){
  response.send("Matty's Home Page");
});
app.get('/users/danny', function(request, response){
  response.send("Danny's Home Page");
});
app.listen(3000,function(){console.log('Express server started')});
```

After initializing this with Node.js, opening the browser with different urls will display differently; i.e. opening <http://localhost:3000/users/ken> will display “Ken’s Home Page”, while opening <http://localhost:3000/users/matty> will display “Matty’s Home Page”. This would work to correctly route any page request, but clearly it isn’t a scalable solution as the server.js file will grow to handle every new page that is created. Instead, we could divide the routing operations, beginning by writing the .js file to handle urls that begin with /users/. In the following demonstration a directory called ‘router’ is created in the root of the project, and then within that directory a file called ‘user_router.js’ created which contains the following code. It has essentially taken the sub-routes from the users part of the previous code. This time an express Router is

created and used to direct the different routes. The final line 'module.exports = router' will allow this router to be used in our next file.

```
const express = require('express');
router = express.Router();
router.get('/ken', function(request,response){
  response.send("Ken's Home Page");
});
router.get('/matty', function(request,response){
  response.send("Matty's Home Page");
});
router.get('/danny', function(request,response){
  response.send("Danny's Home Page");
});
module.exports = router;
```

Next we can edit the server.js file to make use of our new router.

```
const express = require('express');
app = express();
app.use('/users',require('./router/user_router'));
app.get('/login', function(request, response){
  response.send("Login Page");
});
app.listen(3000,function(){console.log('Express server started.')});
```

Now if the path begins with '/users' the new router will handle it. Clearly the same principle could be applied to set up other paths.

15.3 Creating an API with Express.js

An API is an Application Programming Interface and this section investigates how the routing in Express.js can be used to create an API to make interacting with a MongoDB more convenient. The previous chapter created a MongoDB database to store a collection of data about calendar appointments. Now we will create a router to deal with different database requests, such as requesting all appointments, requesting appointments from a particular month, and inserting new data into the database. We begin by setting up a server.js file using express. Whenever a request comes in for localhost:3000/appointments/ they will be directed to the appointments_router.

```
const express = require('express');
app = express();
app.use('/appointments',require('./router/appointments_router'));
app.listen(3000,function(){console.log('Express server started.')});
```

The appointments router can be initially set up as follows.

```

const express = require('express');
api = express.Router();
api.get('/', function(request, response){
  response.send("All");
});
api.get('/:month/:year', function(request, response){
  response.send(request.params);
});
api.post('/', function(request, response){
  response.send("Create");
});
module.exports=api;

```

This example has set up 3 routes. The first is for returning all the appointments in the database, although currently it just returns “All”. The second is to return appointments for a particular month according to the URL, where the request for localhost:3000/appointments/7/2018 would return all appointments in July 2018. Currently this just returns the request parameters and would output as in figure 15.3. The third route is for creating a new appointment, but currently just responds with “Create”. Note that while the first two routes respond to GET requests, the third route responds to a POST request.

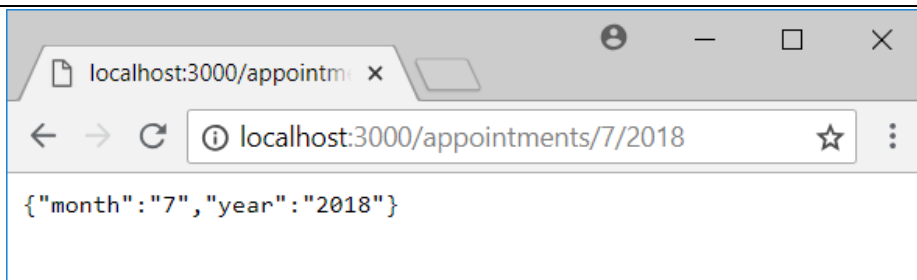


Figure 15.3: Returning Request Parameters

The next step is to replace each of these routes, beginning with the request to return all appointments.

```

api.get('/', function(request, response){
  var MongoClient = require('mongodb').MongoClient;
  MongoClient.connect("mongodb://localhost:27017/calendar", function(err, db) {
    var myDB = db.db('calendar');
    var collection = myDB.collection('appointments');
    collection.find().toArray(function(err, items) {
      response.json(items);
    });
  });
});

```

Assuming the MongoDB database is running and set up as at the end of the previous chapter, this will connect to the database and find all elements in the 'appointments' collection and return them in JSON format, as can be seen in Figure 15.4.

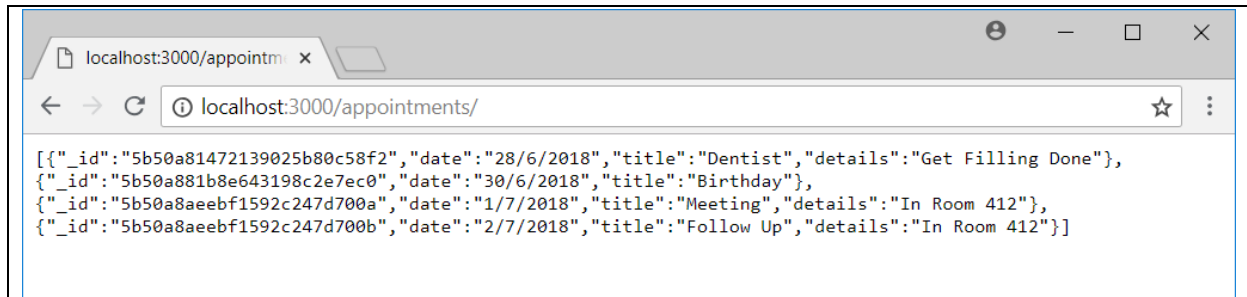


Figure 15.4: API Returning All Appointments

The second route is for dealing with requests for appointments from a specified month and year. It is similar to the previous route, however it requires a regular expression in the query to specify the month and year.

```
api.get('/:month/:year', function(request, response){
  var MongoClient = require('mongodb').MongoClient;
  MongoClient.connect("mongodb://localhost:27017/calendar", function(err, db) {
    var myDB = db.db('calendar');
    var collection = myDB.collection('appointments');
    var re = new RegExp('.*' + request.params.year + '-' + request.params.month + ".*");
    collection.find({date: re}).toArray(function(err, items) {
      response.json(items);
    });
  });
});
```

Once again the results are returned in JSON format, so JavaScript in the browser could be used to display these appointments as required.

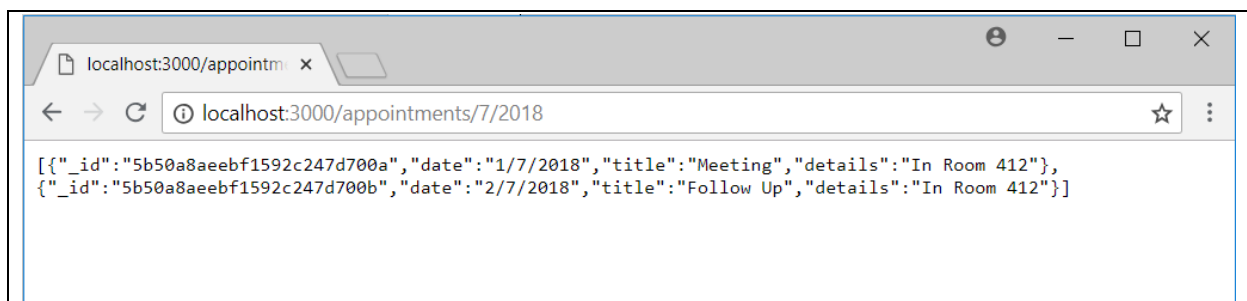


Figure 15.5: API Returning Appointments for Specified Month

The third route is used for creating new appointments and handles POST requests. In order to demo it, we first create a simple form.

```
<form action="http://localhost:3000/appointments/" method="post">
  Date: <input type="date" name="date"><br>
  Title: <input type="text" name="title"><br>
  Details: <input type="text" name="details"><br>
  <input type="submit" value="Submit">
</form>
```

When the form is submitted it sends a post request to the localhost server, including whatever has been filled in. To access the data that has been filled in, another Node.JS module can be used called “body-parser”. First it needs to be installed.

```
C:\www\calendar>npm install body-parser
```

Once the body-parser module is installed, you can require it near the top of the appointments router.

```
var bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
```

Now the body of the request will be parsed, and the contents stored in the request.body object. After submitting the form the request.body object could be examined in the console to see that it is JSON containing the data submitted on the form.

```
{ date: '2018-07-04',
  title: 'Project Deadline',
  details: 'Deadline Day' }
```

Because the fields on the form map neatly onto the fields in the collection, the request.body can simply be inserted into the MongoDB collection. So, a new route can be completed as follows.

```
api.post('/', function(request, response){
  var MongoClient = require('mongodb').MongoClient;
  MongoClient.connect("mongodb://localhost:27017/calendar", function(err, db) {
    var myDB = db.db('calendar');
    var collection = myDB.collection('appointments');
    collection.insert(request.body);
    response.send("Successfully Created");
  });
});
```

Double check the database using compass and you’ll see a new document added to the collection as shown in Figure 15.6. Further routes can also be created and added to the API for other tasks, such as deleting or updating an appointment.

appointments				
	_id ObjectId	date String	title String	details String
1	5b50a81472139025b80c58f2	"2018-06-28"	"Dentist"	"Get Filling Done"
2	5b50a881b8e643198c2e7ec0	"2018-06-30"	"Birthday"	No field
3	5b50a8aeebf1592c247d700a	"2018-07-01"	"Meeting"	"In Room 412"
4	5b50a8aeebf1592c247d700b	"2018-07-02"	"Follow Up"	"In Room 412"
5	5b51fc3a269e9b3ad049000c	"2018-07-04"	"Project Deadline"	"Deadline Day"

Figure 15.6: Result of API to Insert Appointment

15.4 Uploading Files with Express.js

A common task in web development is to upload files to the server, such as allowing a user to upload an avatar, or a gallery of images. In Express.js this can be achieved with the use of middleware. Whilst Express.js is a minimalist framework, it is easily extended by using middleware, as we have already seen with the body-parser module. Middlewares are essentially functions which have access to the request and response objects and are executed 'in the middle', after the incoming request. The 'body-parser' middleware clearly parses the body of an incoming request and creates a new object (req.body) populated with the contents of the body. In this section will look at another middleware called 'multer' which handles multipart/form-data, which can be used for uploading files. The first step, as previously, is to install multer.

```
C:\www>npm install multer
```

Next we can create a simple html file that contains a form. Notice that the form has an enctype property set to "multipart/form-data" as multer won't handle forms that aren't multipart. There is one input field, of type file, with the name avatar.

```
<form action="http://localhost:3000/" method="post" enctype="multipart/form-data">
  File: <input type="file" name="avatar"><br>
  <input type="submit" value="Submit">
</form>
```

This form is set up to upload a single file, although a similar principle can be applied to upload multiple files. The next step is to create a route to handle the file upload, in this case, to keep the example simple, there is just one route for the post method. Before the post method is called, multer needs to set up the destination directory for the file, in this example the destination (or dest) is set to the uploads directory. Details about the uploaded file are logged on the console in the callback function.

```
var express = require('express');
var app = express();
var multer = require('multer');
var upload = multer({ dest: 'uploads/' });
app.post('/', upload.single('avatar'), function (req, res, next) {
  console.log(req.file);
})
```

```
app.listen(3000, () => {
  console.log("Server is listening on port 3000");
});
```

The logged data includes some information about the uploaded file. Note that the original filename has been replaced by a random string.

```
Server is listening on port 3000
{ filename: 'avatar',
  originalname: 'avatar.jpg',
  encoding: '7bit',
  mimetype: 'image/jpeg',
  destination: 'uploads/',
  filename: '3459147af8d62e7ceca948af528974b8',
  path: 'uploads\\3459147af8d62e7ceca948af528974b8',
  size: 10414 }
```

The file will appear in the specified directory, and the information contained in the JSON object can then be stored in a database perhaps with the new filename associated with user's account so the correct avatar could be displayed.

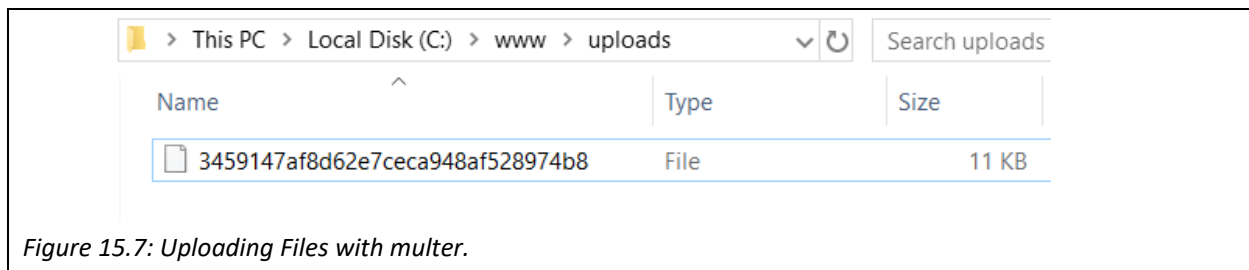


Figure 15.7: Uploading Files with multer.

Key Points

- Express.js is a minimalist framework for Node.js, which means it makes certain tasks such as routing URLs more straightforward.
- Express.js is unopinionated, allowing the developer choice in how to add modules to extend its functionality such as with body-parser and multer.
- Express.js can be used to create a router to handle different Http methods and requests.
- An API can be created to better handle access to data resources, such as in a MongoDB database, a full API will make it possible to retrieve, insert, update and delete database entries.
- Express.js can also be used to handle server tasks such as uploading files.

Further Resources

- 1) The main Express.js website can be found at the following link, complete with a Guide and API reference:
<https://expressjs.com/>

- 2) Tutorialspoint has an introductory guide to Express.js as part of its Node.js tutorial, which can be found here:

https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

- 3) A further tutorial can be found at Tutorialsteacher, here:

<http://www.tutorialsteacher.com/nodejs/expressjs>

Assignment

Extend your MongoDB database for calendar appointments by building an API to make it easy to create, retrieve, update and delete appointments. Add to your create form a field to allow the user to add a file to the appointment which can then be uploaded and a reference to its location be stored in the database.

Chapter 16

Angular

Objectives

Angular is the final part of the MEAN stack, a front-end framework developed by google and written in TypeScript, a superset of JavaScript. The framework is growing in popularity, representing a very different approach to web development than discussed previously, based on a components/services architecture. Angular is particularly popular for single page apps. This chapter introduces the key aspects of Angular and demonstrates a small app. After reading it you should:-

- Be able to set up an Angular app using the Command Line Interface (CLI).
- Understand what components are and how to create and style different components.
- Understand directives such as `*ngFor` and `*ngIf`.
- Understand how data binding works in Angular.
- Be able to use pipes to format output.
- Be able to bind actions to events when the user interacts with the user interface.
- Understand what services are for, particularly how a service can be used to inject data into an app.
- Be able to create a router for use within the browser.

Contents

- 16.1 Getting Started with Angular
- 16.2 Components in Angular
- 16.3 Adding a TypeScript Class
- 16.4 Pipes and Two-Way Data Binding in Angular
- 16.5 Event Binding in Angular
- 16.6 Services in Angular
- 16.7 Routing in Angular

The previous chapter introduced Express.js a web framework that runs on the server, in this chapter we introduce Angular, a web framework that runs in the client. As web development has matured, more good practices have been developed and adopted, including incorporating ideas from standard software development, such as reusable code, object orientation and the use of design patterns such as the observer pattern. We previously introduced Express.js as a server-side framework that works well with Node.js. Express.js is not the only choice though with several others written in alternative languages such as Laravel for PHP, Ruby on Rails for Ruby, Catalyst for Perl and Django for Python. Different developers have different preferences for each, but Express.js is chosen here as part of the MEAN stack, with Angular being the final part. Angular is an open-source, front-end framework based on TypeScript, a superset of JavaScript.

Angular is an open-source, front-end framework based on TypeScript, a superset of JavaScript.

Angular began as AngularJS, developed and maintained by Google, and first released in 2010, however Angular (now) refers to Angular Version 2 (and up). Angular 2.0 was a complete rewrite of the initial AngularJS, first released in September 2016. There remains some confusion between the two and care should be taken when finding supplementary material online to make sure it is referring to the correct version. Before diving into Angular, first we'll briefly explore AngularJS. Written in JavaScript, by Google and a community, with the intention of dealing with some single page applications. There were several frameworks built at this time based on the principles of Model View Controller (or MVC). MVC is an architectural pattern mainly for user interfaces, which focuses on separating the data from the way it is presented to the user, again following a "separation of concerns" principle.

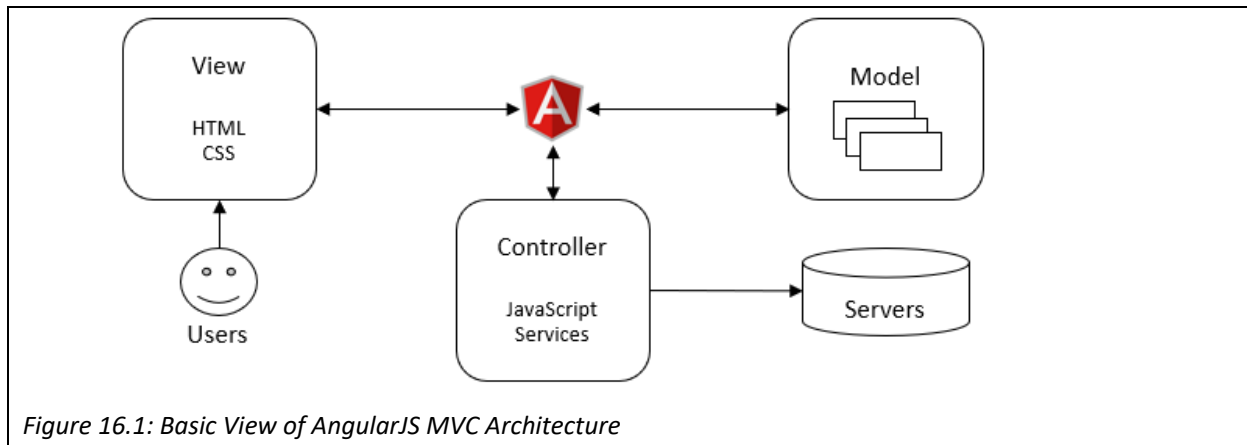


Figure 16.1: Basic View of AngularJS MVC Architecture

The Model refers to the data model, managed separately from the user interface and the way the data is displayed. The View refers to the output representation of the data, and there may be several different views of the same data, for example a table of data could be viewed in spreadsheet form, or as a graph or diagram. The Controller deals with inputs, often validating it, and then passing it on to the model to then update a view. For a blog, each post contains some data, but there could be multiple views of that data – Create, Edit, View,

Preview, etc. AngularJS incorporates MVC into its architecture so the developer can simply follow the good practice, as do many other Front-End MVC Frameworks, such as Kendo, Ember or Meteor. Each framework has its good points and bad points, and each has a community supporting and promoting its development, with differing opinions on which is best. However, AngularJS, developed by Google, and React, developed by Facebook, appear to be the most popular. React describes itself as a JavaScript library for building user interfaces but takes a component-based approach to building a page out of different components. When Angular was rewritten the structure was also based on a components/services architecture, while still taking advantage of the separation of concerns approach from MVC.

16.1 Getting Started with Angular

Getting started with Angular is different from the other languages discussed in this book as first the framework needs to be set up. Angular has a Command Line Interface (CLI) which can be used to create the background files that are needed to get a project up and started. Assuming Node.js is already running, open the console and use the following command to install the Angular CLI globally.

```
C:\>npm install -g @angular/cli
```

Once the CLI is installed, the next step is to create a new project. This can be done again in the command line using “ng new my-app” where my-app is the name of your new project. This may take some time as it creates all the necessary files and directory structure.

```
C:\>md www
C:\>cd www
C:\www>ng new my-app
CREATE my-app/angular.json (3548 bytes)
CREATE my-app/package.json (1310 bytes)
CREATE my-app/README.md (1022 bytes)
```

Once the project is set up, it is time to serve it, or launch the server so it can be viewed in a browser. This is done by navigating to the project folder in the command line and using the ng serve command, adding --open to open a browser to localhost.

```
C:\www\my-app>ng serve --open
```

Opening a browser with localhost to port 4200 will show the default Angular app as shown in Figure 16.2. Meanwhile many files have been created for the project already as can be seen by navigating the my-app directory. Here you will find several files and directories. Firstly a directory called ‘e2e’, which is used for end-to-end tests, and kept separate from the main app as used for testing the complete app. Secondly a folder called ‘node_modules’, where all the third party Node.js modules are stored. The third directory is the ‘src’ directory where your project lives. The rest of the files are configuration files.

Inside the ‘src’ folder a further structure has been created. We shall look into the ‘app’ folder shortly. The ‘assets’ folder is for storing images and other similar assets. The ‘environments’ folder stores files related to different deployment environments as there may be differences between a development or production

environment. The favicon.ico file can be edited to make your site stand out in a bookmark bar. The index.html file is the main project file, written with standard html. Within the <body> tags is an <app-root> tag which calls the rest of the Angular app that we shall write, meaning in reality the main file won't need to be edited. There is also a 'styles.css' file where global stylesheet files can be stored.

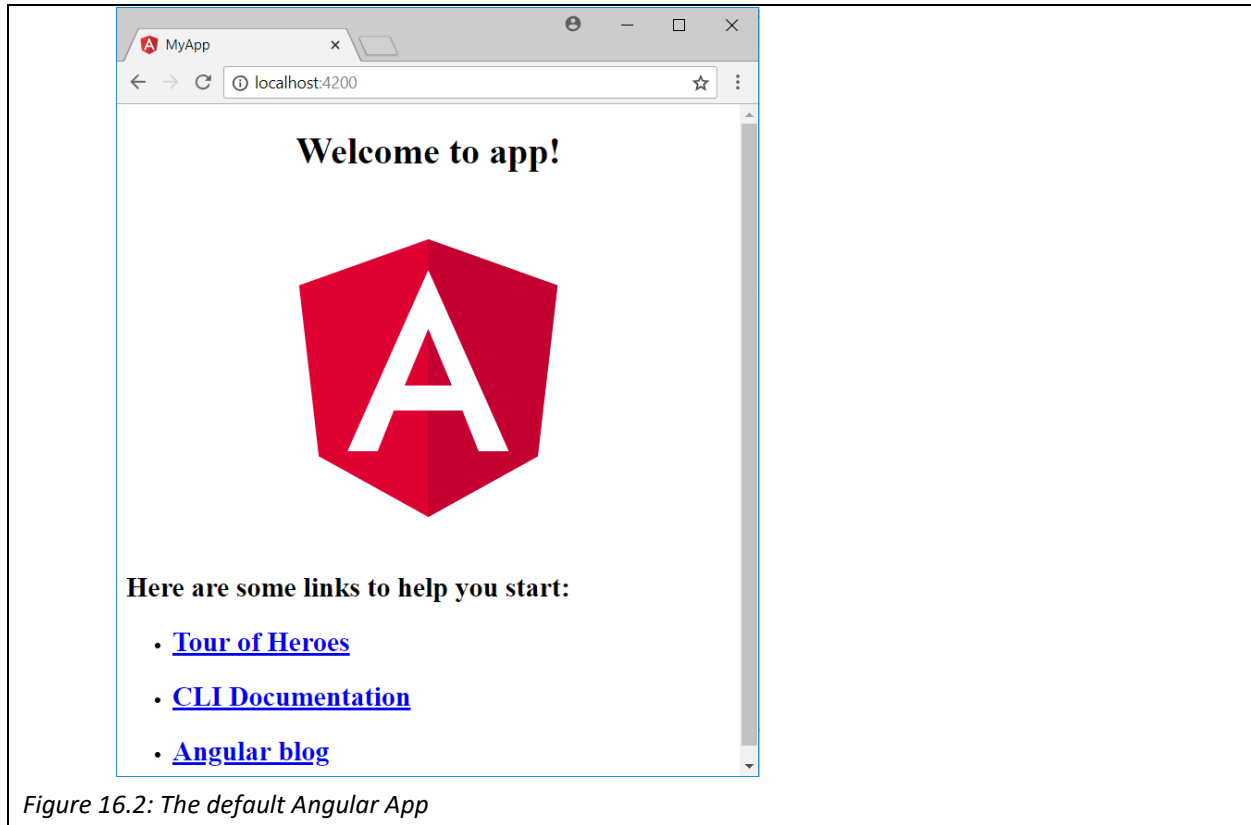


Figure 16.2: The default Angular App

Inside the 'app' directory there are five files already created. We can edit some of these files, beginning with the 'app.component.html' file. This is standard HTML so it should be straight forward to see how this maps onto the default Angular app shown in Figure 16.2. We can edit this file to remove the image and links as follows. Notice that when you save this file, the page is automatically refreshed to display only the title. Also notice {{ title }}, this is the syntax for Angular's interpolation binding system, which binds the component's title value to the property displayed in the header tags.

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1> {{ title }} </h1>
</div>
```

We can change the title element by editing the component class file – 'app.component.ts'. This file contains the code for the main app class, written in TypeScript. While you might not recognize all the code, you should see the similarity with JavaScript. It begins by importing the Component class from the Angular core library, and then decorates it. @Component is a decorator function which adds metadata to the component,

in this case the selector where it should be displayed, the template URL specifying the html file already edited, and the CSS file for its styling.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Hello World';
}
```

Once again, saving the changed title will automatically change the page displayed in the browser window. The third file to edit is the css file 'app.component.css'. Currently this is empty, but here we can specify any styles for the component. Keeping it simple, we could change the color of the text in the <h1> tags.

```
h1{ color: blue; }
```

The result is the creation of a simple "Hello World" in Angular as seen in Figure 16.3.

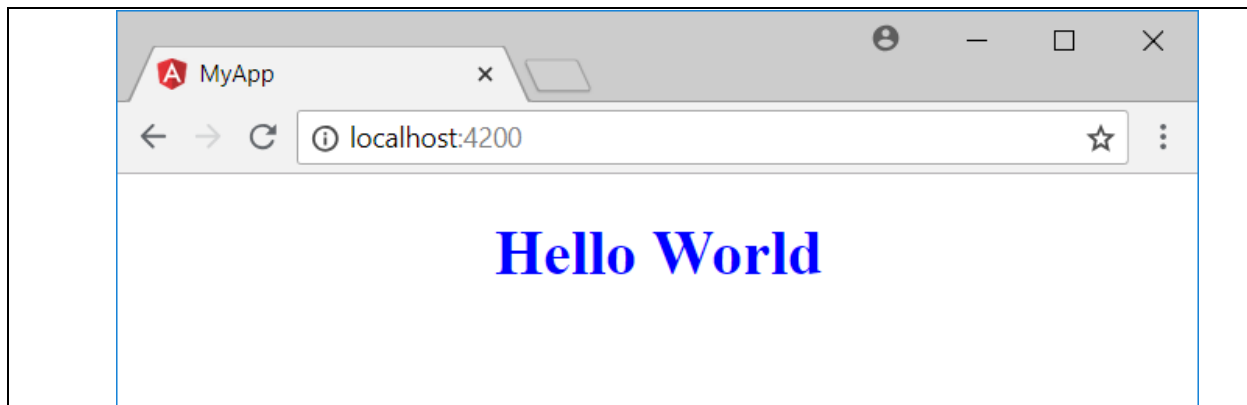


Figure 16.3: Hello World in Angular

16.2 Components in Angular

Components are a vital building blocks for Angular, used for displaying things on the screen, as seen in the hello world example, but also for listening for user inputs and responding to it. Angular is based on a components/services architecture, so it is important to understand how components work in Angular. So far, the AppComponent has created a class for the main app and we have seen how the component separates the template (html), the style (css) and object code (ts). This is the parent component, but further child components can be created to manage different parts of the page. The Angular CLI provides a convenient way to generate a new component as follows.


```
C:\www\my-app>ng generate component team
CREATE src/app/team/team.component.html (23 bytes)
CREATE src/app/team/team.component.spec.ts (614 bytes)
CREATE src/app/team/team.component.ts (261 bytes)
CREATE src/app/team/team.component.css (0 bytes)
UPDATE src/app/app.module.ts (388 bytes)
```

Running the generate component command creates a new directory within the app, and creates 4 files within that directory as well as adding the new component to the main app. Three important files for the component are the class (ts), the template (html) and the style (css). In this example we have created a “team” component and looking in the TypeScript file you will see a similar structure to the AppComponent, i.e. a decorator linking a selector, the template and the style, and an export class. Here the class contains a constructor and ngOnInit(), a lifecycle hook called shortly after the component is created generally used to initialize an object. Within this class we can create a new property, for this example a player with the value “Harry Kane”.

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-team',
  templateUrl: './team.component.html',
  styleUrls: ['./team.component.css']
})
export class TeamComponent implements OnInit {
  player = "Harry Kane";
  constructor() { }
  ngOnInit() { }
}
```

The team.component.html file begins with some brief HTML to indicate that it works, but can be replaced to bind it with our new data as follows;

```
{{player}}
```

Finally, the new component needs to be added to the AppComponent view in ‘app.component.html’, by adding the selector <app-team>.

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1> {{ title }} </h1>
  <app-team></app-team>
</div>
```

For this example, the title of the app has been changed, but as you save your file, the browser automatically updates as in Figure 16.4.

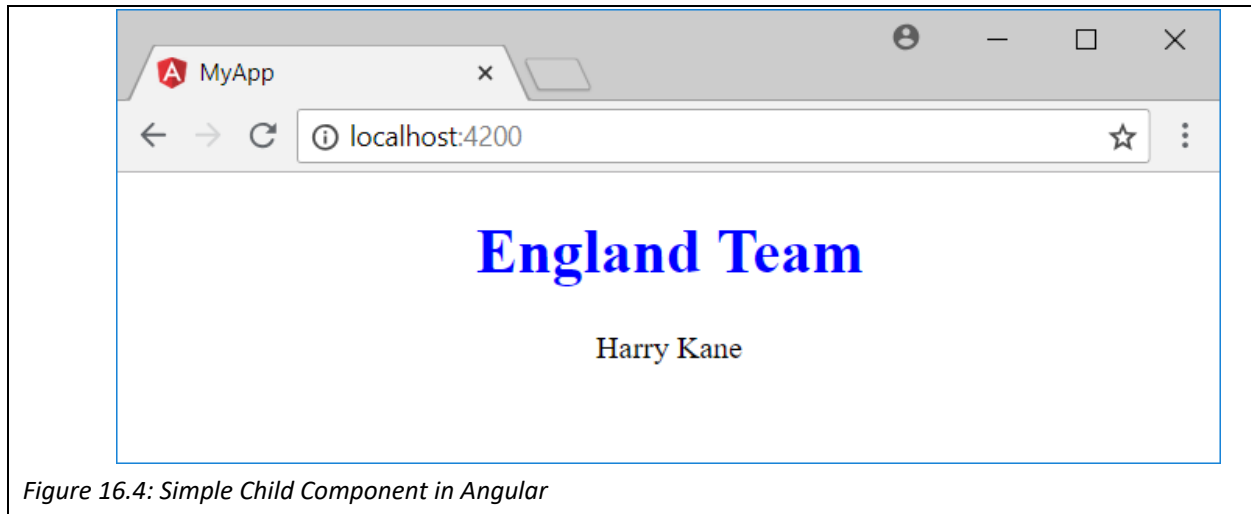


Figure 16.4: Simple Child Component in Angular

So far this has demonstrated a very trivial child component in Angular. This is intentionally so, to keep the code simple, however with the HTML and CSS skills already picked up, it wouldn't be difficult to create components with more interesting templates and styles for different parts of a webpage, such as a header, footer, sidebar, content etc. Rather than making an app look good with HTML & CSS, we shall focus on introducing some more Angular features in the coming sections.

16.3 Adding a TypeScript Class

Currently the player property is just a simple string data type, while in reality this could be a more complex data type including their shirt number, their age, records, image etc. We can create a class for a player in a separate ts file. Within the app directory create a player.ts file and add the following contents.

```
export class Player {
  shirt: number;
  name: string;
}
```

Clearly our Player class is still trivial, but it could easily be extended. Having created a Player class, it needs to be imported into the team component by adding the following.

```
import { Player } from './player';
```

A Player object can then be created within the TeamComponent class. This is only a temporary measure for demonstration purposes as really the data shouldn't be stored here.

```
player: Player = {
  shirt:9,
  name:"Harry Kane",
};
```

The template is currently displaying a string, so needs to be updated to display the relevant parts of the TypeScript object as follows.

```
{{player.shirt}} : {{player.name}}
```

With minimal styling applied, the result is shown in Figure 16.5.

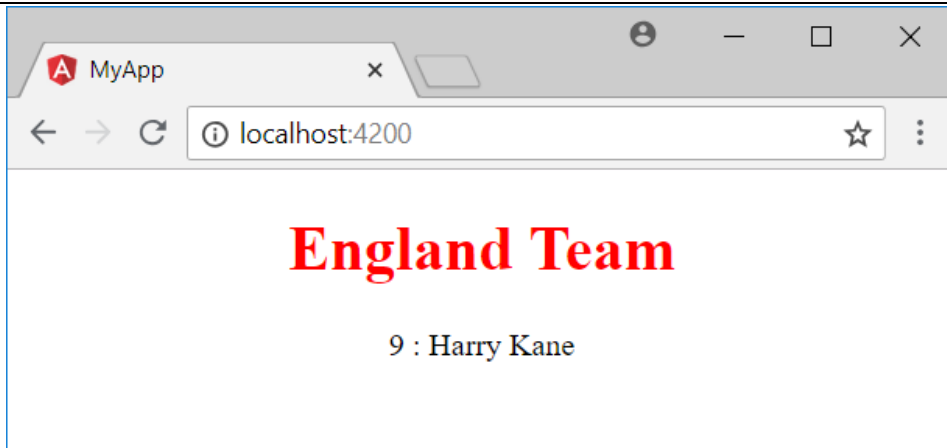


Figure 16.5: Simple Child Component displaying a TypeScript Object

The England team isn't just one man, so the next step is to display all the players, which means invoking multiple player objects and displaying them in the team component. The players can be added to the app in many ways, preferably by accessing a database, but to keep things simple, we can create a TypeScript file to create and store the data in an array within the app directory. In this case we create a file called 'team.ts' with the following sample contents. Note that the file imports the Player class.

```
import { Player } from './player';  
export const PLAYERS: Player[] = [{shirt: 1, name: "Jordan Pickford" }, {shirt: 2, name: "Kyle Walker" }, {shirt:  
3, name: "Danny Rose" }, {shirt: 4, name: "Eric Dier" }, {shirt: 5, name: "John Stone" }, {shirt: 6, name: "Harry  
Maguire" }, {shirt: 7, name: "Jesse Lingard" }, {shirt: 8, name: "Jordan Henderson" }, {shirt: 9, name: "Harry  
Kane" }, {shirt: 10, name: "Raheem Sterling" }, {shirt: 11, name: "Jamie Vardy" }, {shirt: 12, name: "Kieran  
Trippier" }, {shirt: 13, name: "Jack Butland" }, {shirt: 14, name: "Danny Welbeck" }, {shirt: 15, name: "Gary  
Cahill" }, {shirt: 16, name: "Phil Jones" }, {shirt: 17, name: "Fabian Delph" }, {shirt: 18, name: "Ashley Young"  
}, {shirt: 19, name: "Marcus Rashford" }, {shirt: 20, name: "Dele Alli" }, {shirt: 21, name: "Ruben Loftus-  
Cheek" }, {shirt: 22, name: "Trent Alexander-Arnold" }, {shirt: 23, name: "Nick Pope" }];
```

A few more edits are required, firstly to use these players into the Team Component file we need to import the new team.ts file by adding the following line.

```
import { PLAYERS } from './team';
```

Next, we need to replace the player property with our imported team. Here we create a new property called players, removing the data about Harry Kane.

```
players = PLAYERS;
```

Now our team component imports an array of objects of player type. Again, this class could be extended to store more data about each player, such as the URL of their picture or their match rating.

16.3.1 Using the *ngFor Directive

The next step is to update the team component template so that rather than just displaying one player it will loop through our array of players and display each player. As we will display a list of items in a template, it is appropriate to use HTML's and tags. To loop through the array of players we can use Angular's repeater directive "`*ngFor`", which works much like a for loop and is demonstrated below.

```
<ul class="team">
  <li *ngFor="let player of players">
    {{player.shirt}} : {{player.name | uppercase}}
  </li>
</ul>
```



Figure 16.6: Displaying a List Using *ngFor

16.4 Pipes and Two-Way Data Binding in Angular

The previous example also added a Pipe to make the names uppercase. There are several pipes available in Angular, and they are useful for formatting output. Here the uppercase pipe has been applied, but there are others for formatting dates, currencies and so on. It is also possible to create your own pipes. Figure 16.6 shows the results with minimal additional CSS styling.

The next step is to allow the user to edit the team members, and perhaps replace players. For this we can add an input box for each player and use Angular's two-way binding to bind the input box to the objects property. `[(ngModel)]` is the syntax for the two-way binding and the following can be added to the loop to add an input box after each player's name.

```
{{player.shirt}} : {{player.name | uppercase}} <input class="input" [(ngModel)]="player.name">
```

The `ngModel` is an Angular directive and is part of the Forms Module that assists with handling forms and features such as input boxes, but it isn't available by default and needs to be added into the project. There are 2 further steps needed to make it work. First, the `FormsModule` needs to be imported and added to the `app.module.ts` using the following line of code.

```
import { FormsModule } from '@angular/forms';
```

This may be your first time editing the `app.module.ts`, so notice how the `TeamComponent` was already added automatically for you. Secondly, the `@NgModule` imports array needs to add the `FormsModule`, essentially adding any external modules the app needs.

```
imports: [  
  BrowserModule,  
  FormsModule  
]
```

With some basic CSS styling added, the result is as in Figure 16.7. Note that as any textbox is edited, the associated property is also edited.

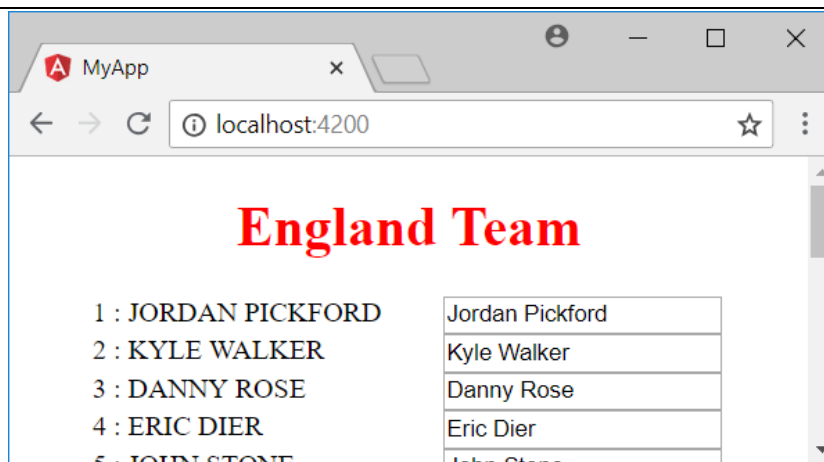


Figure 16.7: Two Way Binding in Angular

16.5 Event Binding in Angular

This section will demonstrate how to bind actions to events, i.e. when the user interacts with the interface. As seen previously in JavaScript there are a variety of ways in which a user can interact with the webpage, so again to keep it simple this example will allow the user to select their captain by using the click event on a player's name. The first thing to do here is to attach a function to handle the click event for any of the players appearing in the list of players.

```
<li *ngFor="let player of players" (click)="onSelect(player)">
```

Here when a list item is clicked upon the 'onSelect()' function will be called, with the selected player sent as a parameter. The function has yet to be written, but should be added to the TeamComponent class. Here we make a couple of changes, firstly adding a 'captain' parameter, and secondly implementing the 'onSelect()' function to assign the captain to whichever player has been selected.

```
captain: Player;  
onSelect(player: Player): void {  
  this.captain = player;  
}
```

Now that the captain has been selected, we should display it on the screen. Of course a new component could be created to display the captain, but we will add it to the existing team component.

16.5.1 Using the *ngIf Directive

Note that when the application is started, there is no captain selected, so we need to add a directive to only display the captain after it has been selected. For this we can use the *ngIf directive as follows.

```
<div *ngIf="captain">Captain: {{captain.name | uppercase}}</div>
```

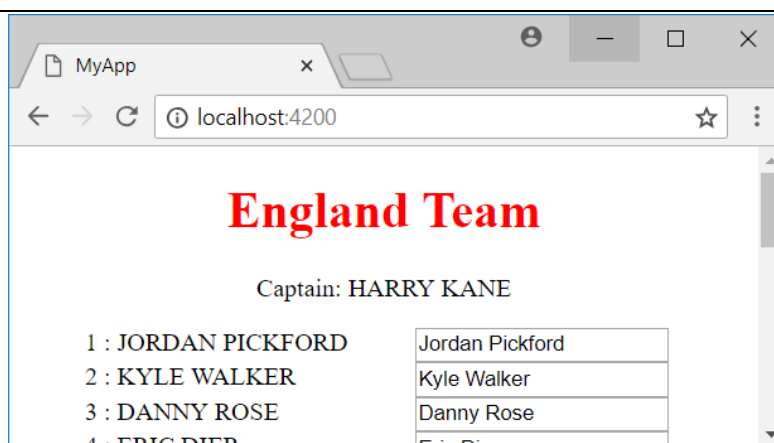
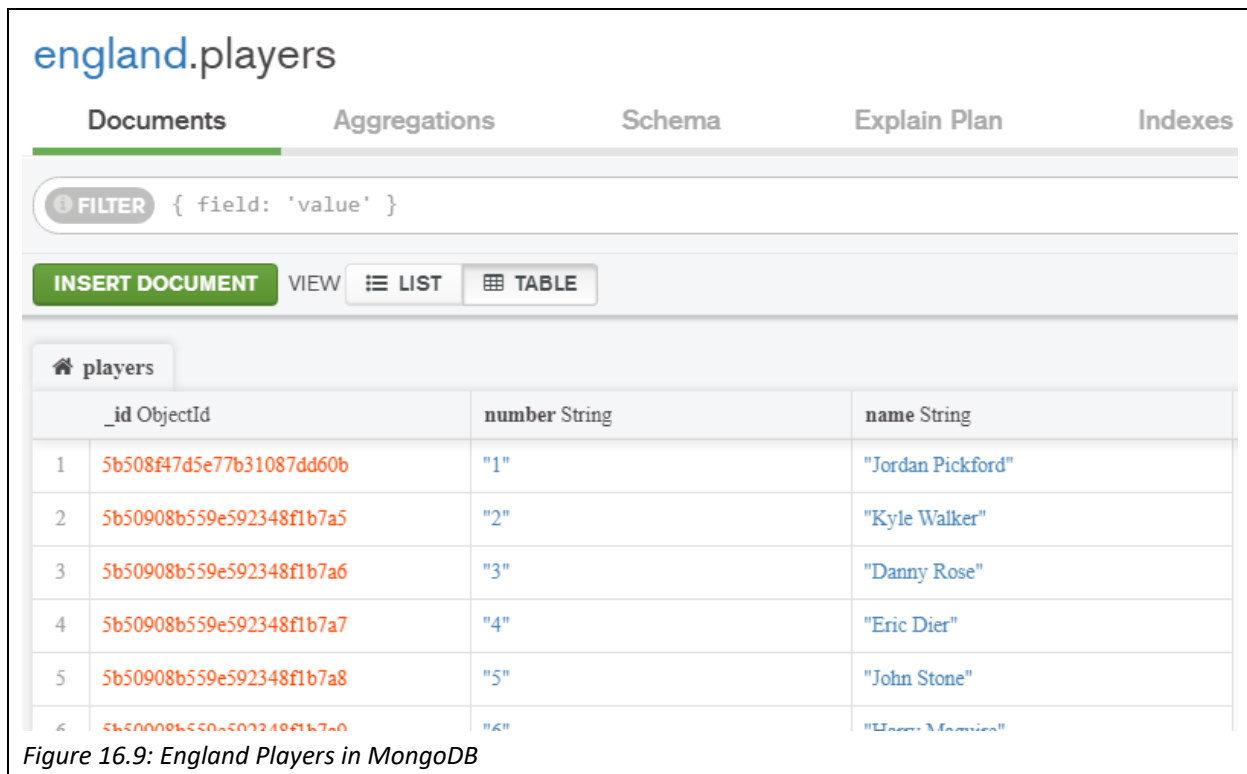


Figure 16.8: Event Binding in Angular

Now when a player is clicked upon they are selected as the captain, and the team sheet is updated to display the captain, again using the uppercase pipe. If no captain is selected then the div doesn't appear.

16.6 Services in Angular

Currently the data for the app is being stored in a file (team.ts), which is then imported into the team component. This isn't ideal for many reasons, not least the challenge of updating data stored in a file format when compared to a database. Also, the component should concentrate on presenting the data and delegate the accessing part of the job to a service. A service can be used when code might be needed in many places, such as when multiple components need to access the same data. In this section we will build a service that uses HTTP to access the team data from a MongoDB collection. Before creating the service, the first step is to set up a MongoDB collection, and create an API to access it, in a similar way to in section 15.3.



	_id ObjectId	number String	name String
1	5b508f47d5e77b31087dd60b	"1"	"Jordan Pickford"
2	5b50908b559e592348f1b7a5	"2"	"Kyle Walker"
3	5b50908b559e592348f1b7a6	"3"	"Danny Rose"
4	5b50908b559e592348f1b7a7	"4"	"Eric Dier"
5	5b50908b559e592348f1b7a8	"5"	"John Stone"
6	5b50908b559e592348f1b7a9	"6"	"Harry Maguire"

Figure 16.9: England Players in MongoDB

After creating the collection, a server can be created node called 'server.js'.

```
const express = require('express');
app = express();
app.use('/',require('./router/team_router'));
app.listen(3000,function(){console.log('Express server started.')});
```

This server directs incoming requests to the team_router, which just deals with two routes. The first route returns the whole collection, while the second route returns a single item based on a specified id. It can be tested by opening the Node.js server, and directing a browser to localhost:3000/ to see the whole collection. Note that this line has been added 'response.header("Access-Control-Allow-Origin", "*");' to allow other connections from localhost.

```

const express = require('express');
api = express.Router();
api.get('/', function(request, response){
  response.header("Access-Control-Allow-Origin", "*");
  var MongoClient = require('mongodb').MongoClient;
  MongoClient.connect("mongodb://localhost:27017/england", function(err, db) {
    var myDB = db.db('england');
    var collection = myDB.collection('players');
    collection.find().toArray(function(err, items) {
      response.json(items);
    });
  });
});
api.get('/:id', function(request, response){
  response.header("Access-Control-Allow-Origin", "*");
  var MongoClient = require('mongodb').MongoClient;
  MongoClient.connect("mongodb://localhost:27017/england", function(err, db) {
    var myDB = db.db('england');
    var collection = myDB.collection('players');
    collection.findOne({number: request.params.id}, function(err, items) {
      response.json(items);
    });
  });
});
module.exports=api;

```

Services are a key part of Angular, kept separate from Angular components. Services can perform a variety of tasks, such as logging errors to the console, validating user inputs, or as in this case fetching data from the server. In this case we will use an HttpClient service to create an instance of a service and inject it into a dependent component. Before we create this service, the HttpClientModule needs to be imported into the app within app.module.ts. Previously we added the FormsModule from '@angular/forms', the HttpClientModule can be found in '@angular/common/http'.

```

import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

```

Also add the HttpClientModule to the imports array in the same file.


```
imports: [  
  BrowserModule,  
  FormsModule,  
  HttpClientModule  
],
```

Previously, we used the Angular CLI to generate a component, and it can now also be used to generate a service.

```
C:\www\my-app>ng generate service team  
CREATE src/app/team.service.spec.ts (362 bytes)  
CREATE src/app/team.service.ts (133 bytes)
```

This creates two files, where team.service.ts contains the TeamService class. Angular automatically creates a skeleton file that we can edit.

```
import { Injectable } from '@angular/core';  
@Injectable({ providedIn: 'root' })  
export class TeamService {  
  constructor() {}  
}
```

This imports and sets up the @Injectable decorator, which is responsible for creating instances of the service and injecting it where it is needed. An injector needs a provider to manage how it is injected, in this case it is provided in root, making it available throughout the application. The role of this service is to construct instances of the TeamService class (using its constructor). In this example the objective is to make available to any subscribers a getTeam() function, which queries the mongoDB collection.

```
import { Injectable } from '@angular/core';  
import { HttpClient } from '@angular/common/http';  
import { Observable } from 'rxjs';  
import { Player } from './player';  
@Injectable({ providedIn: 'root' })  
export class TeamService {  
  constructor(  
    private http: HttpClient,  
  ) {}  
  getTeam (): Observable<Player[]> {  
    return this.http.get<Player[]>('http://localhost:3000/');  
  }  
  getPlayer(id: number): Observable<Player> {  
    return this.http.get<Player>('http://localhost:3000/'+id);  
  }  
}
```

We need to extend this service in several ways – firstly by importing HttpClient so we can make Http requests. The constructor for the service is then used to create an http object which is in turn used in the getTeam() method. The new getTeam() method is used to make the default query via express to return an array of Observable Players. The getPlayer() method will be used later to return a single player.

Now that the service is set up, the next step is to edit the team component to make use of TeamService. This requires a few edits to the TeamComponent, beginning by importing the new TeamService. An instance of the TeamService is declared in the constructor, and then a getTeam() method implemented to subscribe to the TeamService injector. The method is then called in ngOnInit(), shortly after the component is constructed.

```
import { Component, OnInit } from '@angular/core';
import { Player } from '../player';
import { TeamService } from '../team.service';
@Component({
  selector: 'app-team',
  templateUrl: './team.component.html',
  styleUrls: ['./team.component.css']
})
export class TeamComponent implements OnInit {
  constructor(private teamService: TeamService) {}
  ngOnInit() {
    this.getTeam();
  }
  players: Player[];
  captain: Player;
  onSelect(player: Player): void {
    this.captain = player;
  }
  getTeam(): void {
    this.teamService.getTeam()
      .subscribe(players => this.players = players);
  }
}
```

A minor edit is needed to the template file and now the result is much the same as after the previous section, with the key difference that now the data is loaded from the database. Using a similar approach further routes can be created to create, update or delete records from the mongoDB.

```

<div *ngIf="captain">Captain: {{captain.name | uppercase}}</div>
<ul class="team">
  <li *ngFor="let player of players" (click)="onSelect(player)">
    <div class="row">
      <div class="col">{{player.number}} : {{player.name | uppercase}}</div> <div class="col"><input
class="input" [(ngModel)]="player.name"></div>
    </div>
  </li>
</ul>

```

16.7 Routing in Angular

The previous chapter dealt with how Express is used to handle routing on the server, in a similar way Angular can be used to handle routing on the client. To demonstrate this, we will need to create a new component to show the details of each player, but first we can use the Angular CLI to generate an app-routing module. In this command a couple of modifiers are added; --flat instructs the module to be placed in the app directory rather than in its own directory, and --module=app registers it within the imports array of the AppModule.

```

C:\www\my-app>ng generate module app-routing --flat --module=app
CREATE src/app/app-routing.module.spec.ts (308 bytes)
CREATE src/app/app-routing.module.ts (194 bytes)
UPDATE src/app/app.module.ts (623 bytes)

```

The key created file is the app-routing.module.ts file. To begin with we can add a couple of routes to our existing TeamComponent. Note that the key changes are to include the TeamComponent, and then to set two routes.

```

import { NgModule }      from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { TeamComponent }  from './team/team.component';
const routes: Routes = [
  { path: '', redirectTo: '/team', pathMatch: 'full' },
  { path: 'team', component: TeamComponent }
];
@NgModule({
  imports: [ RouterModule.forRoot(routes) ],
  exports: [ RouterModule ]
})
export class AppRoutingModule {}

```

The intended result is for <http://localhost:4200/team> to open our existing TeamComponent, while <http://localhost:4200> will be redirected to the same team page. Before this will happen we need to add the

<router-outlet> element to the main App Component Template (app.component.html). For this example the app will display the title, and then whichever route is directed to the router-outlet tags.

```
<div style="text-align:center">
  <h1> {{ title }} </h1>
  <router-outlet></router-outlet>
</div>
```

Testing it out should point both URLs to display the teamlist. To demonstrate routing more effectively we need to create a new component, where the details of each player will be displayed. While no further details will be added, the database could easily be extended to store more data such as a photo, ratings etc. For this example the “captain” feature is removed, as well as the onSelect() event, to be replaced with a link to the new component. The Angular CLI can be used to create a new component called ‘player’.

```
import { Component, OnInit } from '@angular/core';
import { Player } from '../player';
import { ActivatedRoute } from '@angular/router';
import { TeamService } from '../team.service';
@Component({
  selector: 'app-player',
  templateUrl: './player.component.html',
  styleUrls: ['./player.component.css']
})
export class PlayerComponent implements OnInit {
  constructor(
    private route: ActivatedRoute,
    private teamService: TeamService
  ) {}
  ngOnInit(): void {
    this.getPlayer();
  }
  player: Player;
  getPlayer(): void {
    const id = +this.route.snapshot.paramMap.get('id');
    this.teamService.getPlayer(id)
      .subscribe(player => this.player = player);
  }
}
```

The player component is similar to the team component, except that it uses the getPlayer(id) method from TeamService, rather than getTeam(). In order to populate the id parameter, Angular’s ActivatedRoute is used. The template file adds a navigation link back to the default router, and the input box complete with two-

way binding is moved from the teamsheet to the player's details. Of course a further route would need to be added to update the database if the player's name is changed.

```
<nav>
  <a routerLink="/">Display Team</a>
</nav>
<div *ngIf="player">Number: {{player.number}} Name: {{player.name | uppercase}}
<br>
<input class="input" [(ngModel)]="player.name">
</div>
```

Finally a further path is added to the app-routing.module.ts file.

```
{ path: 'player/:id', component: PlayerComponent }
```

The result is two different routes, where the main route shows the teamlist as in Figure 16.10.

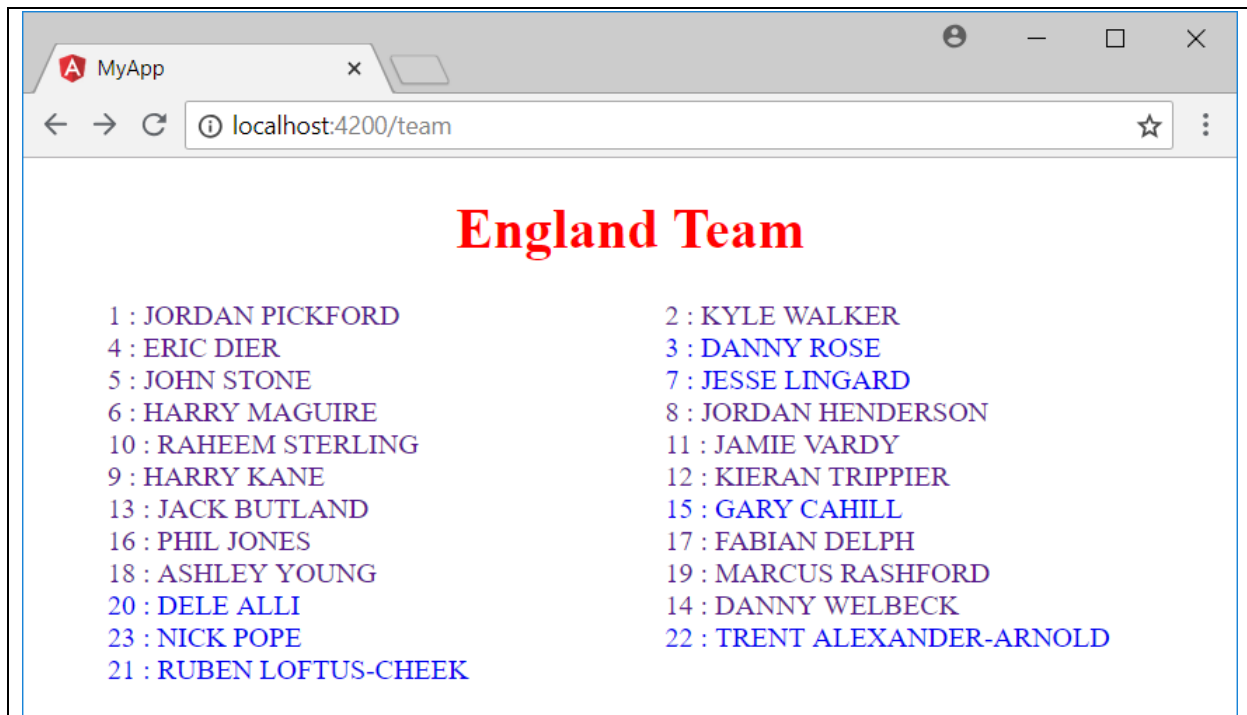


Figure 16.10: England Team List View

The second displays details of which ever player is selected as in Figure 16.11.

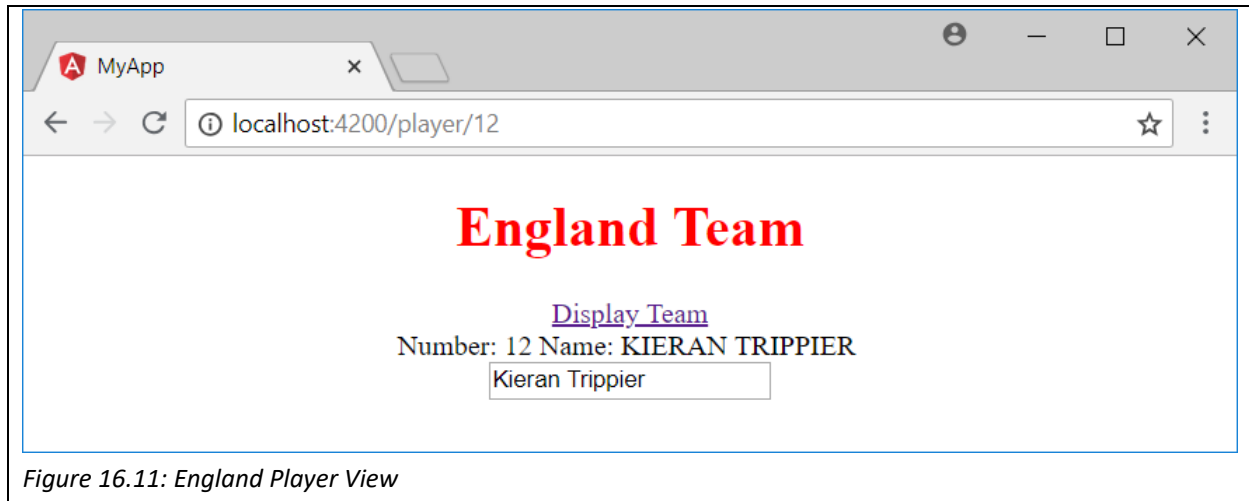


Figure 16.11: England Player View

Key Points

- Angular is a front-end framework based on the component/services architecture.
- Angular is significantly different from AngularJS.
- Angular is managed via the Command Line Interface (CLI).
- Components are vital building blocks in Angular, used to display different parts of the screen.
- Each component consists of an object (written in TypeScript), a template (written in HTML) and a stylesheet (written in CSS).
- The `*ngFor` directive can be used to loop through an array, while the `*ngIf` directive works like an If statement.
- Pipes are used to format output in different ways.
- Services are used to separate concerns in development, for example separating the presentation logic from the data, so data can later be injected into the app.
- The `HttpClientModule` can be used as a service to request data via Http from a server to inject into an app.
- Just as a server can handle different requests via different routes, Angular manages routing within the browser to display different components for different URL requests.

Further Resources

- 1) The main Angular website can be found at the following link.
<https://angular.io/>
- 2) Angular provides an introductory tutorial here:
<https://angular.io/tutorial>
- 3) Tutorialspoint has a tutorial on Angular4, which was released in March 2017, here:
https://www.tutorialspoint.com/angular4/angular4_overview.htm

Assignment

Now you have all the tools to be set free on web development, and there are so many directions you can go – it really depends why you wanted to learn web development. By now you should have a calendar written in PHP, MySQL and JavaScript – why not repeat the project using the MEAN stack?

Another project could be to write blogging software, allowing a user to create, view, edit posts, without needing to understand any code.

This is similar to a Content Management System, which allows a user manage the content that is displayed on their website, again without needing to understand the code.

The web is vital for Electronic Commerce, so platforms are needed for merchants to display their products and allow users to find products to buy.

A social network is a way for people to connect and share posts with each other, with features such as chatting.

This book has given you a foundation from which you can build. If you look at existing software, you should now be able to understand how it works and extend it, or if you prefer, you should be ready to begin one of these projects from scratch.

Index

A

<a>	9
Absolute Link	8
Access Permissions	80
ActiveObject	157
AJAX	156
ajax()	160
Alt	9
Angular	218
Animation Effects	133
Apache	45, 184
API	211
Append	78
<app-root>	221
Arrays (JavaScript)	113
Arrays (PHP)	48
Array Functions (PHP)	59
Arrow Operator (PHP)	71
<article>	170
Associative Arrays (JavaScript)	114
Associative Arrays (PHP)	48
Attributes	8
attr()	142
Audio	172
Auto_Increment	83

B

	5
Background-color	23
Background-image	23
Background-position	24
Background-repeat	24
Binding	221
blur()	125
<body>	4
body-parser	214
Boolean	48
Borders	35, 144
Box Model	36, 144
 	6
Browser Events	129
Brute Force Attack	100
Buttons	53

C

Callback	133
Canvas	173
CAPTCHA	100
Cascading Style Sheets	12, 222
charset	169
children()	148
chmod()	80
Class	15
Class Selectors	17
Clearing Floats	35
click()	126
Collection	200
Color	21
Comments (HTML)	10
Comments (PHP)	47
Compass	201
Components	222
Console	109
Context Sensitive Selectors	21
Cookies	93, 179
\$_COOKIE	94
Cross Site Scripting	104
D	
Database	82, 195
Date Picker	161
Dates (PHP)	66
dblclick()	126
DELETE (MySQL)	86
<details>	170
Dependency Injection	231
Dialog Box	163
Doctype	169
Document Object Model (DOM)	31, 117, 138
DOM Traversal	150
<div>	19
Do...While Statement (JavaScript)	113
Do...While Statement (PHP)	51
Draggable	164
Dreamweaver	3,19
Droppable	164
Drupal	184
E	
echo	46
	5
Email	192
Encryption	98
eq()	152

Escape Character	110
Events	125, 189, 228
explode()	48
Express.js	207

F

Fade Effects	130
Favicon.ico	221
fetch_array()	89
fetch_assoc()	89
fetch_object()	90
Files (PHP)	78
File Modes	78
File System	188
File Upload	215
filter()	152
Filters	150
find()	149, 202
findOne()	203
first()	151
Flash	172
Floating Page Elements	33
Floating Point Numbers	48
focus()	125
	14
Font-family	22
Font-size	23
Font-variant	23
Font-weight	23
<footer>	170
For Statement (PHP)	51
For Statement (JavaScript)	112
Foreach Statement (PHP)	52
Forms	53, 118
Form Events	128
Framework	208
Functions (JavaScript)	115
Functions (PHP)	68

G

Geolocation	177
GET	209
get()	159
\$_GET	54
getElementById()	116, 123
Gradients	176
Grandparents	146

H

<h1>...<h6>	5
<head>	4
<header>	170
hide()	124
Hover pseudo-class	39
HTML	3
HTML Selectors	17
html()	141
HTML5	169
HTTP	93, 186
Hypertext	9

I

<i>	5
IDs	10
ID Selectors	18
If Statement (JavaScript)	111
If Statement (PHP)	50
Images	8
	8
implode()	48
Include	69
innerwidth()	144
Injection (Dependency)	231
INSERT (MySQL)	85
Integer	47

J

JavaScript	107
jQuery	122
jQuery UI	160
JSON	109, 156, 185

K

Key=>Value Pair	48
Keyboard Events	127

L

LAMP	45, 184
last()	152
lcfirst()	48
Letter-spacing	22

	5
Line-height	22
Links	8,9
Links Array	115
Links & CSS	24
Lists & CSS	25
load()	158
Localhost	46
Local Storage	178

M

MAMP	45, 184
Margins	36, 144
Mathematical Functions (PHP)	64
MEAN Stack	183
Method (Forms)	53
Modules	186
MongoDB	199
Mouse Events	126
MVC	184, 214
MySQL	45, 81, 195
MySQL Workbench	196
MySQLi	87

N

<nav>	170
Nested Tags	6
*ngIf	228
*ngFor	226
Node.js	183
not()	153
Notepad	3
Node Package Manager (NPM)	190

O

Objects (PHP)	70
Operators (JavaScript)	110
Operators (MongoDB)	204
Operators (PHP)	49
outerwidth()	144

P

<p>	5
Padding	35, 144
Parameters (HTML Tags)	8

Parameters (JavaScript)	116
parent()	145
parents()	147
parentsUntil()	147
Passwords	55
PHP	44
PHP String Functions	48
PHP Operators	49
Phpmysqladmin	82, 196
Pipes	227
Position:Absolute	18,30
Position:Fixed	30
Position:Relative	29
Position:Static	29
POST	209
post()	159
\$_POST	53
Prepared Statements	100
print_r()	54,60
Pseudo-class	24
R	
Radio Buttons	119
Random Numbers (PHP)	66
React	220
readyState	157
Relative Link	8
Require	69
responseTxt	159
Responsive Design	37
Routing	209, 233
S	
<script>	107
<section>	170
SELECT (MySQL)	84, 198
Selectors	16
Services	229
Sessions	95
Session Hijacking	104
\$_SESSION	95
Semantic Tags	170
SHA (Secure Hash Algorithm)	98
Slide Effects	132
Sorting (PHP)	63
	19
SQL - Structured Query Language	81
SQL Injections	100, 103
statusTxt	159

Strings (PHP)	47
Strings (JavaScript)	109
String Functions (PHP)	47, 48
	5
Sublime Text	3
submit()	129
<summary>	170
Superglobal Array	53
Switch Statement (JavaScript)	112
Switch Statement (PHP)	50

T

Tables (HTML)	7
Tables (MySQL)	83
<table>	8
<tbody>	8
<td>	8
Text	22
Text-align	22
Text-decoration	22
Text direction	22
Text-transform	22
Text-indent	22
Text Manipulation (jQuery)	140
<tfoot>	8
<thead>	8
Time (PHP)	66
<title>	4
Topstyle Lite	19
<tr>	8
Traversing the DOM	145
TypeScript	221

U

	5
UPDATE (MySQL)	85
url	187
UTF-8	169

V

val()	141
Varchar	83
Variable (JavaScript)	108
Variables (PHP)	47
Video	172

W

WAMP	45, 184
Wampserver	45
Web Workers	180
WHERE (MySQL)	86
While Statement (JavaScript)	113
While Statement (PHP)	51
width()	144
Wordpress	184
Word-spacing	22

X

xhr	159
XML	156
XMLHttpRequest	156

Z

Z-index	18
---------	----

Glossary

AJAX (Asynchronous JavaScript and XML) – A JavaScript programming technique which allows the browser to send data to, and read data from the server to update part of a page, after it has loaded.

Angular – A Front-end framework based on TypeScript developed mostly by a team from Google.

Apache – An open-source webserver.

API (Application Programming Interface) – A set of methods to facilitate communication between components of a program.

Array – A simple data structure for managing a collection of elements.

Browser – A software application used for viewing pages across the World Wide Web.

Callback – A function that is sent as a parameter to another function, such that it is executed after the function has finished executing.

Client – The user's machine, which sends requests for services offered by the server machine.

CSS (Cascading Style Sheets) – The language which defines the style of HTML elements, which tells the browser how HTML elements should be displayed.

Cookie – A small piece of data sent by a website and stored on the client's computer while a user is browsing a site.

DOM (Document Object Model) – A programming API for HTML documents, defining the logical structure of elements on a page.

Express.js – A web application framework for Node.js, designed for building applications and APIs.

Framework – A generic set of functionality which can be selectively changed or extended by the developer.

Geolocation – An API for locating a users position, so that location based services can be offered.

GET – A commonly used HTTP Request method used to get data from a server.

HTML (Hyper Text Markup Language) – A core language of the web, used to mark up text by adding a variety of tags to it.

HTTP (Hyper Text Transfer Protocol) – The core protocol for exchanging hypertext between different machines.

Hyper Text – Structured text which uses links (hyperlinks) to connect to other pieces of hypertext.

JavaScript – A core programming language used in web development. All browsers support JavaScript, so initially it was the key language for client-side scripting, while today JavaScript is run in many places, including on some servers.

jQuery – A library of JavaScript functionality which makes writing JavaScript code simpler, particularly for event handling, DOM manipulations and AJAX.

JSON (JavaScript Object Notation) – A human readable notation for JavaScript objects which is used for transmitting data objects as well as viewing them.

Localhost – a hostname referring to the same computer, used to make a client machine act like different server machine.

MongoDB – a document-oriented database program, which stores data as JSON-like documents.

MVC (Model View Controller) – An architectural pattern for developing user interfaces, where the problem is broken down into 3 parts; a model, or the data, views that dictate how the data should be displayed, and a controller which responds to user inputs.

MySQL – A relational database management system which uses Structured Query Language to store data in tables.

Node.js – A JavaScript run time environment that enables JavaScript to be used on the server, rather than just the browser, so that JavaScript code can be run before a page is sent to the browser.

PHP (Hypertext Pre Processor) – A server side scripting language used to generate hypertext before sending it to the browser.

POST – A commonly used HTTP Request method used to send data to a server.

SQL (Structured Query Language) – A language for managing data in a relational database. It allows structured statements to create, read, update and delete items in the database.

TypeScript – A typed superset of JavaScript designed for developing large complex applications, which then transcompiles into JavaScript.

URL (Uniform Resource Locator) – A web address, or reference to the location of a resource on a computer network.