Web development has grown to become vital in shaping how humans interact, work, learn, and consume. Practical Web Development provides you with a roadmap of web development today, giving you the tools you need and the guidance to keep you creative and productive in a world of disruption and innovation.

Beginning with the structure of the Web and the principles of building basic websites with HTML, you will learn about CSS, JavaScript, and PHP, before taking a closer look at some of the leading technologies used to build the modern Web. You will integrate jQuery, Ajax, and JSON into your projects before moving on to the latest tools and techniques in responsive web design, including Zurb Foundation or Bootstrap, to help you meet the challenges of developing for multiple devices, and explore how Node.js offers a powerful solution to server-side application development. This book is for anyone that wants to get to grips with the broader picture of web development today.

Who this book is written for
This book is perfect for beginners who want to get started and learn the web development basics, but also offers experienced developers a web development roadmap that will help them to extend their capabilities.

What you will learn from this book
- Find out how HTML lays the foundation of web development
- Learn the fundamentals of web design using CSS
- Harness JavaScript to bring websites to life
- Use PHP and MySQL to dynamically generate HTML on the server
- Learn how to write compact code with jQuery
- Create efficient single page applications with Ajax
- Discover JSON for a powerful data exchange between client and server
- Design Mobile first, responsive pages that look great on mobiles, tablets, and desktops
- Create and migrate powerful and scalable web applications with Node.js

Learn CSS, JavaScript, PHP, and more with this vital guide to modern web development

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'The World Wide Web'
- A synopsis of the book’s content
- More information on Practical Web Development
About the Author

Paul Wellens has been a senior product manager for a major computer company in the Los Angeles area and Silicon Valley for over two decades. Before that, he used to install Unix systems and train companies in Europe, from his native Belgium. Later in his career, he became a web development aficionado because it brought him back to another passion of his: programming.

This is not his first book. His prior publication is of a different nature. Nature is what it is all about as it is a guidebook on Eastern California, which is illustrated with his own photographs. Therefore, it should not come as a surprise to learn that, besides experimenting with new web technologies, his major hobbies are photography and hiking in the Eastern Sierra.
Preface

I am fortunate to have lived and worked in California for a long time. The majority of that time, I lived in Palo Alto, which is the center of Silicon Valley, the home of Stanford University, and the birthplace of many companies, big and small, such as Sun Microsystems, where I worked. I sat on the front row to see how the World Wide Web developed, as well as being present for the advent of social media. Facebook started on the other side of the wall of my favorite restaurant. Now, some Facebook guy or girl is sitting in what used to be my office at the bottom of the Dumbarton Bridge. As a product manager for Solaris, one of my tasks was to make sure that Netscape Navigator was included with our operating system. So, I was right at the source in which the development of the Web began. I even went to the Web 2.0 conference and bought the book of the same name.

Then, I felt the need to have my own website to display my photographs and inform people about the beauty and interesting places of the parts of California that I had discovered during my many journeys travelling around the state. So, I created one. One day, I was telling a friend about it and he tried to look at it on his mobile phone. It looked terrible. So, I bought a Nokia phone (a brick compared to what we have today) so that I could test my own site to make sure that it looked good on a phone as well. This is how I caught the bug of responsive design, years before someone started calling it this.

Upon my return to Belgium, I decided that it was time to learn as much as possible (I love to learn new things) about what is out there beyond creating websites and took a 6-month course on PHP web development. A lot of it looked familiar as I was previously a UNIX and C developer. There were only 12 people in the class, who were all bright minds, and I quickly discovered that there was more to learn.
Preface

As the classes took place in Leuven, a major university town in Belgium, I went to the local university bookstore and bought book after book on all kinds of related topics and quickly became a jQuery fan. jQuery, by the way, was not even included in the course. I started wondering why someone needed to have 35 different books to learn about web development and that writing a single book that gave a comprehensive overview of what you need to know to engage in web development would not be a bad idea.

Since then, web development has changed a lot; more books were needed, eBooks this time, but the concept remained the same. So, now you know why I wrote the book.

This book gives you an overview of all the general aspects of web development, in a traditional way, using plain HTML to do static websites, as well as the current way, to enable you to create your web pages dynamically and make sure that they look great on mobile devices as well, by using responsive design. We conclude by giving you a hint of what is yet to come if you replace the traditional web server by writing your own using node.js.

What this book covers


Chapter 2, *HTML*, introduces HTML and gives you an overview of the most commonly used HTML tags to do web development. You will be able to create a basic website after reading this chapter.

Chapter 3, *CSS*, explains how to use Cascading Style Sheets (CSS). This is used for the presentation part or layout of your website, from color to dimensions to typefaces. The most commonly used CSS properties are explained here. Once you are done with this chapter, you will be able to make your basic website look good.

Chapter 4, *JavaScript*, first gives you an introduction to the world of programming and programming languages. Next, the overall syntax of JavaScript and how to use it for client-side programming is introduced.

Chapter 5, *PHP*, explains PHP, which is another programming language. This one is used to do server-side programming. It requires a web server to do the development of your website and deploy it. You will learn how to dynamically create your web pages, rather than having to write a bunch of HTML files.
Chapter 6, *PHP and MySQL*, introduces MySQL, an open source database. You will learn how to create a database, manage it using the phpMyAdmin tool, and perform basic CRUD (create, replace, update, delete) operations from within a PHP program.

Chapter 7, *jQuery*, covers a popular JavaScript library. It allows you to write more compact and clean code and handles browser incompatibilities for you. With this, it is going to be a lot easier and faster for you to write JavaScript code that traverses and manipulates the web page. It does so by using selectors, which you learned to use with CSS. So, with jQuery, you can write JavaScript code without having to learn a lot of JavaScript.

Chapter 8, *Ajax*, introduces Ajax. It represents a collection of techniques to make it easy to dynamically change only portions of a website. With this chapter, we have entered the world of what I call "modern web development". The interface that we use for our Ajax calls is jQuery.

Chapter 9, *The History API – Not Forgetting Where We Are*, explains a very important piece of the web development puzzle. Once we are changing pages on the fly so they look different but actually remain the same page (URL), strange things can happen when visitors want to go back to what they think is the previous page. A solution for this is described here that will not only work for HTML5 but for HTML4 as well.

Chapter 10, *XML and JSON*, describes XML and JSON. They are two popular formats to exchange data, for example the server and the client. Although XML is used in a variety of environments, JSON is closer to the web development community.

Chapter 11, *MongoDB*, describes an alternative to MySQL as a database. This is a so-called NoSQL database and a document database. Documents are conveniently in the JSON format. Here, how to access a MongoDB database from within a PHP program is described.

Chapter 12, *Mobile First, Responsive Design with Progressive Enhancement*, has the longest chapter title of the book. It explains how modern web development has to be done now that more people are using mobile devices instead of traditional computer screens to go to websites.

Chapter 13, *Foundation – A Responsive CSS/JavaScript Framework*, describes most of the features of the Foundation framework, which helps you with your responsive design. It contains everything that I have always wanted to write myself but never had the time to do. This concludes the part of the book that covers what I call modern web development.
Chapter 14, Node.js, gives an overview of what I call the avant-garde of web development. It introduces node.js, which allows you to write your server-side code in JavaScript, including your own web server, which is facilitated by using the Express framework.

Appendix, Bootstrap – An Alternative to Foundation, describes the popular CSS/JavaScript framework, which is an alternative to Foundation to help you with responsive design. The main reason to include this is to point out key differences and similarities.

The online chapter, The Mono County Site, provides a full example of a website or application where we apply most, if not all, the things we learned. It is available at https://www.packtpub.com/sites/default/files/downloads/B03816_Appendix.pdf.
The World Wide Web

This book talks about the past, present, and future of Web Development. Beginning with Chapter 2, HTML, we will walk you through all the technologies you need to know about, in order to practice web development. Before we do that, we want to set the stage, so that we all know which Web we are talking about: this would be the World Wide Web (www).

World Wide Web

I love history! So let us start with a little history about the World Wide Web.

I was fortunate enough to be able to work at a company that developed the first commercial version of the UNIX Operating System. They were founded in 1977 and I joined them ten years later. UNIX is an Operating System (the thing you need to make your computer do something) that was intended to run on minicomputers (although they were called that, they could not fit into your apartment and required air-cooling). These computers were typically used as an isolated system that had quite a number of text-based terminals attached to them.

Today UNIX lives on, and forms the basis of Linux, Solaris, MacOS, and others. Our company spotted an opportunity to add products that would add features and technologies that today are standard. Some of these examples are email (ability to send a mail to a person on another computer), and ftp (ability to transfer a file to another computer, or to just access another computer). Yes, you had to pay extra if you wanted to be able to send mail. Thanks to the Internet, all of this was made possible.
The Internet

The Internet is a global network that today interconnects billions of computers worldwide. Its origin dates back to research done for the US government, but today everybody can get on the Internet, using its standard set of protocols, commonly referred to as TCP/IP (the IP here being Internet Protocol).

Every computer or device that is connected to the Internet will have a unique Internet address, aka IP address. It is a set of 4 numbers separated by dots, for example, 192.25.13.90. Of course, you will never tell your friend that you bought something at 192.25.13.90 but at, for instance, www.amazon.com. This is because the Internet also uses a feature that translates IP addresses into easier to remember domain names. The example I used happens to be a website you can go to and buy things from a practice that we all know has caused the www to become what it is today.

The Internet and the www are, in everyday speech, treated as if they are synonyms but they are indeed not. There are a lot of different services provided on the Internet (by companies called Internet Service Providers (ISP)), and that was already the case before the www existed (for example, giving you access to electronic mail). What it took for the www to emerge was (just like in JavaScript) a series of asynchronous events. Two of those are too important to not mention.

HTTP and HTML

You have probably heard this story many times, but the World Wide Web would not have existed without it. It is the story of Tim Berners Lee, an engineer at the European Centre for Nuclear Research (CERN). The centre had many computers that were, of course, hooked up to the Internet. It also produced a tremendous amount of data and documents, and that became almost impossible to manage. Tim worked out a solution by developing a language to write these documents in, a protocol on top of the Internet to manage them, as well as a computer program for users to access them.

HTML

HTML, short for Hypertext Markup Language is the name of that language. Hypertext is text that contains hyperlinks, which in turn are those parts of a document which readers can click on to take them to a different document, using the link. You have all seen the blue underlined parts of a text, in not-so-good-looking web pages. These are hyperlinks.
A document in HTML consists of tags, with text in between them. There are opening and closing tags for example, as follows:

```html
<h1>Hello, world</h1>
```

Here, `<h1>` is the opening tag and `</h1>` the closing one. We will learn about a similar markup language: XML. HTML and XML are not the same though. One important difference is that in XML you can define your own tags, as long as you close each one you’ve opened. XML is used to transfer the data and the tags are used to organize the data.

In HTML, tags do have a specific meaning. `<h1>` would be used in a document for the text of a level one header. A `<a>` tag—the anchor tag—is the one used to include the hyperlinks we just discussed. So the purpose of writing HTML is not to transfer data, but to present it to human users.

To do so, these tags are interpreted by the computer program we mentioned earlier. Such a program is called a browser. When the reader clicks on a hyperlink, the browser will detect that as well, and send a request to yet another program, the web server, to go fetch another document.

### HTTP

This is where HTTP, the Hypertext Transfer Protocol fits in. If a user clicks on a link, it is like saying: go fetch another HTML document. The name of that document would be part of a longer string that starts with `http://` and also contains the domain name of the server. It is called a uniform resource locator, but we all refer to it as URL. Following is an example: `http://www.paulpwellens.com/examples/secondpage.html`.

What you can do with HTTP has evolved over time and we will learn about it later on in the book, but for now we need to move on with our history lesson. One more little tidbit of history for you: guess how our friend Tim called his browser, the first ever browser: WorldWideWeb. He later renamed it to avoid confusion.

### The World Wide Web Consortium (W3C)

After he left the CERN in 1994, Tim Berners Lee founded the World Wide Web Consortium (W3C). The consortium tries to enforce compatibility and agreement between vendors that deliver components for the web. Incompatible versions of HTML would cause browsers to render web pages differently; and incompatible features added to browsers have the same unexpected result.
If you visit www.w3.org, the consortiums website, you will notice that the W3C has evolved into a standards body for many technologies, but even in those days, having such an organization was sorely needed.

**Mosaic**

History moved on, and so did I! Our company was acquired by Sun Microsystems and I became product manager for our PC UNIX product. Oh yes, before I forget, PCs were everywhere by then and that phenomenon would, in the long run, also contribute to the explosion of the World Wide Web.

One day, in the year 1993, my engineering manager walked into my office, together with Jonathan, his lead programmer. They wanted to show me what he (Jonathan) had done over the weekend. It was a port (take the source code of a program and make it run as a binary on a computer) of a program called Mosaic for our PC UNIX product. I saw him type a few commands but did not quite understand why these guys were so excited. Little did I know that this seemingly innocent little program was going to change our lives forever!

**The first browser**

Mosaic was developed at the National Center for Supercomputer Applications (NCSA) at the University of Illinois in Champaign-Urbana (this is a long drive through cornfields from Chicago, which I took once) by a team led by Marc Andreesen. It was the first browser to support multiple protocols (hence the name) as well as display an image and text on the same page (surprisingly, this is not a trivial matter on a webpage!) It soon caught attention worldwide and the browser was ported to many platforms, so that more and more people could develop or have access to websites.

In November 1992, there were 26 websites. In the Mosaic browser, there was a What's new section that showed a new website everyday. Three years later, there were 10,000 and another three years later, millions. Today, I do not think it is possible to count them anymore. So how did we get from thousands to millions?

**Netscape**

Marc Andreesen founded a company, named it after Mosaic, and then later renamed it Netscape Communications Corporation. They basically rewrote the Mosaic browser and optimized it for environments with lower network bandwidth, such as individuals who access the Internet from home through their ISP. The browser was called Netscape Navigator. This was clearly the first commercial browser, making it to the shelves of computer retail stores as part of a bundle, Netscape Communicator.
Netscape was also credited as the first browser to include support for JavaScript. With this scripting language, interactivity could be added to web pages. The World Wide Web could be at anybody’s fingertips, as long as you had Netscape. From 1994 to about 1999, Netscape clearly had the biggest market share for browsers.

Over time, Netscape was moved over to Mozilla Corporation, an Open Source organization, and the development of Netscape ended. Today, users can download the Firefox browser from www.mozilla.org.

**Internet Explorer**

Another derivative of Mosaic, Spyglass Mosaic, made it into the Microsoft codebase and was eventually bundled with Microsoft Windows. We know it today as Internet Explorer. This is how we reach the topic of the so-called browser wars. As I mentioned, Netscape was trying to win over as many customers as possible by getting into the retail market and charge for its browser. Microsoft, on the other hand, decided to bundle Internet Explorer for free in its Windows Operating System (of course you had to pay for Windows).

This led to many lawsuit-like situations as Microsoft was accused of unfair competition by the other browser vendors. This situation has now changed as all browsers are free. Computer users can today choose which browser they want to use. On tablets, choices are limited, but on the other hand, browsers on tablets usually have all the latest features.

A different and potentially more bloody war was going on at the technical level. Despite having a standards body, the W3C, which controlled and introduced new features (both HTML and CSS), not all browsers were adapting those features equally as fast. So the same web pages continued to look different when they were rendered by different browsers. Sad but true to say, the browser that was the most incompatible and unpredictable, Internet Explorer, was, by the turn of the century, also the one used by the majority of the people who would surf the web.

Developers therefore had no choice but to delay the use of new cool features at the expense of spending a considerable amount of extra time making their web pages look the same on a PC used by most visitors as it did on the system where they were created.

Things got far worse before they began to get better, as more developers started using JavaScript to add interaction and animation to the pages, whereas many System Administrators recommended a configuration with JavaScript switched off. Sometimes this resulted in the visitor seeing nothing at all on the page.
But do not despair, we are in 2015 now! In this book, we will take a different approach and always let you use the new features when the browser supports it.

The explosion of the Web

By the turn of the century, every company wanted to have a webpage. Web pages were created by linking more web pages, with information about the company or just the owner of the site. The latter was made possible by smart ISPs that also offered web hosting. People have to be able to access your site even while you or your computer is sleeping; so these services offer 24/7 uptime to put your HTML files. Web hosting companies also take care of getting you a domain name, such as www.thecoolestphotographer.com.

At some point, obtaining the domain name you wanted was bordering on another browser war, as there could only be one xyz.com, and if some entrepreneurial folks thought that having xyz.com first would be worth money, they would grab it.

When I wanted one for myself, paulwellens.com was already taken by a British rugby player, so I went for www.paulpwellens.com (P is my middle initial). I am neither British nor a rugby player so this was fine with me.

So a lot of pages were created worldwide, but all they had initially was information for you to look at, nothing else. In some cases, they were created once and never updated. That fortunately was the exception to confirm the rule. Many companies decided that they had to have a presence on the web and corporate websites were thus created. The advent of CSS facilitated this a lot, as it allowed the separation of presentation and content. That way, the marketing department would provide the corporate logo, and the look and feel, and all the other departments would provide the content.

Amazon.com and e-commerce

Some creative minds realized that the Web presented an opportunity to do more than just provide information. It is only a (relatively) small step from providing the information of the products you have on your site, to actually selling them. E-commerce was thus born. Amazon comes to mind as a good example of a site that everybody can relate to as being a Web shop. Developing a web shop of course involved a lot more than having someone in the company type in some content.
These products are real products; they sit in a warehouse, have a part number, a price, a name and description, different sizes and colors, and so on. That information, more than likely, is already present in some database that is updated each time a product is sold in a retail store. To sell something online, your webpage has to interact with the visitor, present him with some kind of an on-screen shopping cart, calculate subtotals, check warehouse inventory, and so on.

To do this, more and more programming was involved, and not just data entry into an HTML file anymore. So the job of Web Developer was born. Traditional programmers are skilled in a single programming language (Java or C++) and usually a single platform (Solaris or .NET). Web Developers have to be fluent in at least four different languages, as well as know a thing or two about databases. I would like to add one aspect which, to me, makes this job very exciting—one gets to be involved in design aspects. The gap between a Web Designer and a Web Developer is narrowing.

So today, one talks about Front-End and Back-End developers.

So this is what we will teach you in this book; how to be a Web Developer, but not before wrapping up our history lesson. There are a few more things that made the web what it is today.

**Google and Yahoo!**

So you have a website with information, or a web shop because you are selling something on the web, like [www.mycoolproduct.com](http://www.mycoolproduct.com). How do you expect to reach your potential customers, call every single one of them? This is where Google or Yahoo fit in. These popular companies developed the so-called search engines.

You want to find out everything there is to know about a movie you just watched, a song you cannot remember the name of, or simply the phone number of your favorite restaurant? You visit [google.com](http://google.com), [yahoo.com](http://yahoo.com), or equivalent sites and type in what you are looking for. Chances are you will find it.

We did a usability study at work where we gave one group a set of CDs, a stack of manuals, and a computer to install; the other group did not get the manuals but Internet access. The second group did way better because they felt they did not need the manuals, as they assumed them to probably be out of date, and immediately looked things up online. And this happened over 10 years ago.

Today, the use of search engines is so commonplace that terms like Yahoo! and Google are used as verbs. In some languages, they actually have become verbs and made it into the official dictionary.
Social networking

I know people who do not use Facebook today, but I do not know anybody who does not know of Facebook. For years I walked by the building where Facebook had its office. I was not really interested. Then, when I moved back to Belgium, I decided to join them so I could stay in touch with my friends in California, who live in a different time zone and several thousands of miles away. From one of them I found out that the Facebook folks have since moved into the building where I used to have my office. Funny how that goes sometimes!

Facebook, Twitter, YouTube, and LinkedIn are examples of popular social networking sites. Nothing is sold here, but shared. People share pictures, stories, events, thoughts, ideas, opinions, and so on.

Web development

Many years ago, I took a 6 month class on what, basically, is part one of this book. Months into it, it became apparent that the lack of an introductory part that explains how all the components of the course were related, was the course's biggest flaw.

After six months, there were still people who did not understand the difference between Java and JavaScript. So I promised myself two things: that one day I would write a book, and that such a chapter would be part of it. So let's go!

HTML

Files written in HTML form the basis of every website. We briefly touched on its history in the previous section; here we will dig a little deeper in its structure. Look at the following example:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Hello World example</title>
</head>
<body>
  <h1>Hello, world</h1>
</body>
</html>
```

The first line specifies DOCTYPE, referring to the HTML version used, so the browser knows how to interpret the file. The one in the example is indicative of HTML5. DOCTYPE statements used to be a lot longer.
This is followed by the main tag, the `<html>` tag. In-between, we will find all our HTML in two sections, `<head>` and `<body>`. The body tag is what contains your content, and the head tag contains other information. In our example, there is one metatag that specifies what encoding is used. The `<title>` tag contains text that will be displayed by the browser at the very top of the window. It is very important to not forget the `<title>` tag, as this is one of the things search engines will examine.

In this simple example, the body contains a single `<h1>` tag. This represents a level one header in the document, similar to headers you find in word processors. The browser will decide how to display that content or, as they say, render it. So how do we get the HTML into a file and how do we get it to a browser?

**HTML editors and other tools**

As an HTML file is just a text file, your favorite text editor will do just fine. Just make sure it has the `.html` extension in the name, for example `hello.html`.

However, at some point, you are probably going to include some CSS, JavaScript, and surely PHP in the same file, in which case specialized tools will make you far more productive.

**Browsers and web servers**

So now you have a file called `hello.html` and you want to look at it in a browser. In real life, this file will be part of your website and you will have to put it there. This is where the company that is hosting your website told you to put your files. They will give you all the information to correctly transfer your file(s) to their server.

They will end up in a folder that is called the **document root**, the root of all the files that make up your site. If you were to follow the instructions with the `hello.html` file and transfer it there, you will see the result when you type the following address in the URL bar of your browser:

```
http://www.mycoolsite.com/hello
```

You can also look at your file locally, and we will teach you more about that in the next chapter.

It is very important to realize that to the people who visit your site, your web page may not look the same as what you created. One factor—but not the only factor—is the browser that is being used. We therefore recommend that, from the early development stage on, you look at your work using different browsers and increase the number of browsers, for/and different devices.
Always install Mozilla Firefox and Google Chrome on your Mac or PC. Pick one to do your development (I like Firefox because of Firebug), but always do a little bit of testing with other browsers before you deliver.

So, start with our little example, and you will see that even Hello World will look different in different browsers. Fortunately, we can control almost all of this by using CSS.

**CSS**

Cascading Style Sheets (CSS) is a technology that works nicely in accordance with HTML and allows you, not the browser, to determine what your page will look like.

Look at this slightly modified example of our Hello, World webpage, hello.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Hello World example</title>
  <link href="hello.css" rel="stylesheet"></link>
</head>
<body>
  <h1 class="green header" id="hello">Hello, World</h1>
</body>
</html>
```

In the line containing the `<h1>` tag, we added two HTML attributes, `class` and `id`, to the `<h1>` element. Attributes are strings inside an HTML opening tag of the format `name="value"` and the two most important attributes you can use are `class` and `id`. Many elements can be part of several classes, but `ids` are unique to a single element.

Now create a file called `hello.css` with the following content:

```css
h1 {
    font-family:Baskerville, cambria, serif;
    font-size:24px;
}
.green {
    color:green;
}
#hello {
    font-weight:bold;
    font-style: italic;
}
```
This is our first CSS stylesheet. The first rule in the file means that any <h1> element in our document will be in the Baskerville font (or cambria, if Baskerville is not present on the user's computer), at 24 pixels, and in the color the browser has chosen (usually black).

However, when it, or any other element, not just h1, has a class="green" (in CSS, the . in name means class name), it will be displayed in the color green.

Finally, our specific Hello World header will be displayed in bold and italics because of the last few lines in the CSS file. The # character is used in CSS to indicate an identifier, so the #hello rule means a rule for the element that is set to the id= "hello".

As a result of all of this, any browser should render our HTML file as a line containing the text Hello, world, displayed in letter type Baskerville (a serif font often used for eBooks, no relation to Sherlock Holmes that I know of), in green, of size 24 pixels, and in bold and italic. Just try it, it works!

Note that we did not repeat the Baskerville line in the #hello rule, the rule is simply inherited. The <h1> rules cascade into #hello, as this happens to be an <h1>, hence the name Cascading Style Sheets. As we just demonstrated, we can clearly separate the content and the presentation of our page by using CSS. That is why it is important to learn how to use CSS as early as possible.

So, as a Web Developer, you already know that you need to master at least HTML and CSS. We will now move on to the next piece of the language puzzle—JavaScript.

**JavaScript**

When we talk about JavaScript in this book, unless noted otherwise, we mean client-side JavaScript. All the code is interpreted, just like the HTML and CSS, by the browser.

By using JavaScript, we can add action to our pages and interaction with the visitors of our website, as well as change the contents and look of our page through programming. Let us take a look at the following example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Hello World example</title>
  <link href="hello.css" rel="stylesheet"></link>
</head>
<body>
</body>
```
If you look at this page in a browser, there will be no **Hello, World** displayed on your screen, but a pop-up box will appear with a question. If you answer the question **Yes**, our familiar green **Hello World** text will be back. The pop-up box itself will look completely different when you use a different browser.

If you look at the code, you will recognize programming-like stuff. There is an `if` clause and there is a variable (answer). Note that the name of the variable is a normal character string but in its declaration it is preceded by `var`. All of the JavaScript code is in-between an HTML `<script>` tag with a `type` attribute of `text/javascript`.

There is one line that is very typical to JavaScript and does all the work for us:

```html
    document.getElementById("hello").innerHTML="Hello, world";
```

*Chapter 4, JavaScript* is where we will really teach you what this means. For now, we will give you the English interpretation of this line of code: In our document, replace the inner content of the HTML tag with `id` hello by the string **Hello, world**.

In subsequent chapters, we will introduce JavaScript libraries, which will allow you to write more compact JavaScript code, with a lot of work already done for you. **jQuery** is one of these libraries and will be discussed in *Chapter 7, jQuery*.

**PHP**

JavaScript is a complete language and allows you to do a lot more things than those we showed you in the previous little example. However, as I mentioned, this is client-side JavaScript, interpreted by the browser. So once you switch off your computer or tablet, it is all gone. Well, some of it may be saved on your machine.
Imagine trying to create an online store using only the languages we have mentioned so far. That would not work. The information of what is available in the store, as well as the data of your specific order has to live somewhere else. That would be the computer of the company that runs the store, not the device that runs the browser used to visit the site.

So, dear Web Developer, you have guessed it, you will have to learn at least one more programming language to deal with all of this, before you can create an online store. The language itself could be any of several (it can even be JavaScript), but where the code is stored and interpreted is the key difference here: a remote Application Server. One of the most popular of these languages is PHP, which is covered in detail in Chapter 5, *PHP*. Let us look at the following example:

```php
$hello = "Hello World Example";
$helloheader = '<h1 class="green" id="hello">Hello, World</h1>';

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title><?php echo $hello; ?></title>
<link href="hello.css" type="text/css" rel="stylesheet"></link>
</head>
<body>
<?php
  echo $helloheader;
?>
</body>
</html>
```

So far we have been able to test our little examples in a browser, but this is different. To try out this example, you will need to have an Application Server, local or not, installed. For now, just read on.

Notice the <?php and ?> strings in the example. This is the beginning and end of where the PHP code resides and will have to be interpreted by that Application Server. The first portion of the code defines two variables. Note that in PHP, names of variables start with a $ sign, whereas in JavaScript they don't. echo, familiar to UNIX folks, simply echoes the value of these variables.
So once the AppServer is done interpreting the PHP code, all you are left with is our HTML example from the CSS section. This is exactly how it works: the AppServer interprets the PHP code, and then the WebServer passes the resulting HTML code to the browser.

**Apache** is the name of a very popular AppServer that happens to be a WebServer at the same time. This is software that runs on a computer we also call a server and this is where your program file resides: `hello.php`.

So `http://www.mycoolsite.com/hello` will be, once again, the way this webpage can be accessed. This may look like a little bit of using an overkill tour an additional language to display **Hello, World**. But there are some situations where you'd want to do so, for example if the data you need in your HTML is stored somewhere else.

**Data**

One of the main reasons to use the remote server and the server-side language is going to be the storage and manipulation of data. This data can be in several formats, from a flat text file to a spreadsheet, XML, JSON, or a full-fledged database, which requires a Database Server. In the latter case, you may need to learn yet another language, **Standard Query Language (SQL)** and deal with another (software) server: a database server. We will address several options in this book.

**Summary**

In this chapter we discussed the advent and history of the World Wide Web. Next we discussed Web Development in general with a few examples. In order to become a developer of Web Applications, you will have to master at least four languages: HTML, CSS, JavaScript, and a server-side language such as PHP.

Depending on how you plan to organize your data, there may be a need to learn a fifth language (SQL) as well. There will also be more things to learn, such as how to use a library or framework. The good news is that they all have their role in the overall picture of what a Web Application is all about.

Now that we know what we need to learn, let us go do it! We will start with HTML.
Where to buy this book

You can buy Practical Web Development from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.