



## TASK

# Getting Started with Your Bootcamp and HTML

[Visit our website](#)

## WELCOME TO THE TASK ON GETTING STARTED WITH YOUR BOOTCAMP AND HTML!

Well done on making one of the most important decisions of your life. Moving into a tech career can, at times, feel alienating, but we are here to walk with you throughout your programming journey with us.

To kick-start this journey, we are going to introduce you to the command line, using package managers, setting up your development environment, and computer programming. We are going to introduce you to a workflow that will not only serve you throughout this bootcamp but for the rest of your life.

Additionally, you'll learn HTML, the standard language for creating web pages. HTML skills provide an essential web development foundation. Let's dive right in!

## The command line and terminal

As a web developer, it is essential to use libraries and frameworks to be able to build applications. However, to do this, it becomes important to familiarise yourself with the command line. The command line is a tool that you will often use as a web developer. You will use the command line for many subsequent tasks.

### WHAT IS THE COMMAND LINE AND WHY DO YOU NEED IT?

The command line is a means of interacting with a computer program where the user issues commands to the program in the form of successive lines of text. With the command line, you can quickly issue instructions to your computer getting it to do precisely what you want it to do. The command line has rarely been used by most end users since the advent of the graphical user interface (a more visual way of interacting with a computer using items such as windows, icons, menus, etc.).

For web development, you will find it helpful to use the command line when interacting with your files, especially those created using frameworks and libraries. You will also need to be familiar with the command line to work with version control systems like Git. Hence, this task will allow you to acquaint yourself with some of the basics of the command line.

## FINDING THE COMMAND LINE



In Windows, you can simply click the Start menu and type **cmd** in the search box to locate the command line. Alternatively, the command line should be one of the options under 'Programs' and you can simply click on the application to open it.



With Mac OS, open the command line by opening the terminal. This can be done by opening the Applications folder, navigating to Utilities and then launching Terminal. Alternatively, you can search for "terminal" to find the application to launch. (Yang, 2015)

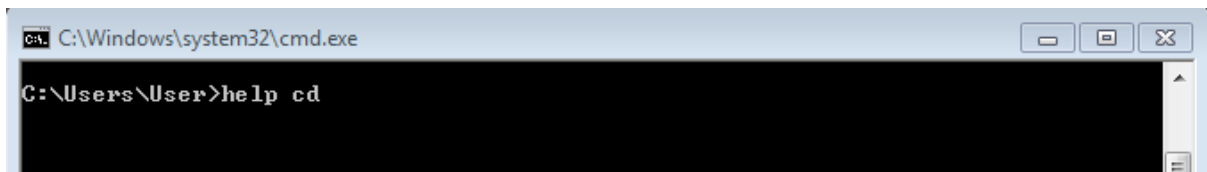
## COMMON COMMANDS

The following table provides a selection of commonly used PowerShell and Unix commands to help get you started with the command line:

Description	Windows cmd	Windows Powershell (alias)	macOS/Linux
Displays the current working directory	chdir	pwd	pwd
Changes the directory	cd	cd	cd
Move up one level in the directory	cd ..	cd ..	cd ..
Displays a list of a directories files and subfolders	dir	dir	ls
Print contents of a text file	type	type	cat
Create a new directory	mkdir	mkdir	mkdir
Remove files and directories	del / rmdir	del / rmdir	rm
Move or rename files and directories	move / ren / rd	move / ren	mv
Copy files and directories	copy	copy	cp
Clear the screen	cls	cls	clear
Quit the terminal	exit	exit	exit

For a comprehensive list of commands, visit [Powershell \(Windows\) commands](#), [Command Prompt \(Windows\) commands](#) and [Unix \(macOS/Linux\) commands](#).

The command line has the built-in **help** (Windows) or **man** (Mac OS/Linux) command. This can be used to view all the executable commands. At this point, why not type the **help/man** command into the command line of your computer and hit Enter to find out more about all the commands? To get help on a specific command, you have to type **help** followed by the command in Windows:




```
C:\Windows\system32\cmd.exe
C:\Users\User>help cd
```

Or type **man** followed by the command in Mac OS/Linux:



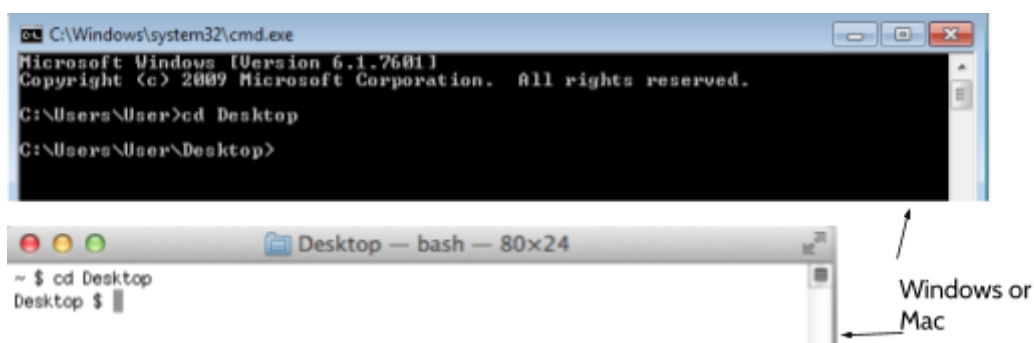
```
~ $ man cd
```

You could also type **whatis** followed by the command in Mac OS/Linux to get help. Compare the output you get with the **whatis** command with the output from the **man** command:



```
~ $ whatis cd
```

The command (in the images above) will give you the information about the **cd** command. As will be noted by the information provided by the command line, the **cd** command is used for navigation. It takes you from one directory to the next. For example, say you want to perform some command on a folder that is on your Desktop, you would have to type **cd** to change the directory to your Desktop as shown in the images below:



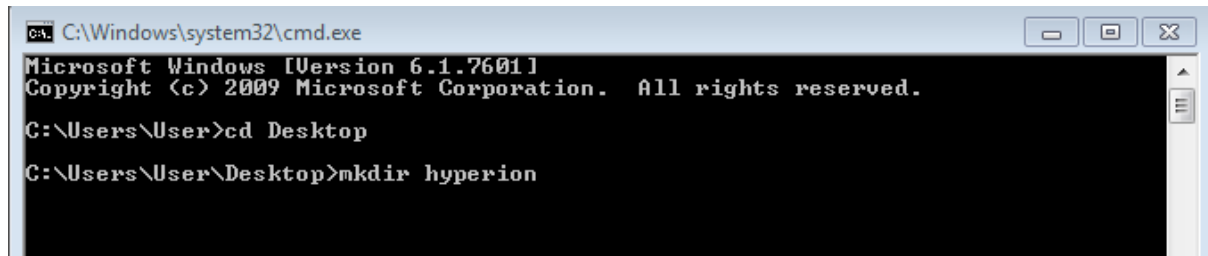
```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\User>cd Desktop
C:\Users\User\Desktop>

Desktop — bash — 80x24
~ $ cd Desktop
Desktop $
```

Windows or Mac

From here, we can now perform operations on the files or folders in our Desktop, since we have navigated into it. But, what if we have forgotten the name of the file or folder that we wanted to operate on? Well, you can simply use the **dir** (Windows) or **ls** (macOS/Linux) command to get a list of all the files or folders saved on the Desktop.

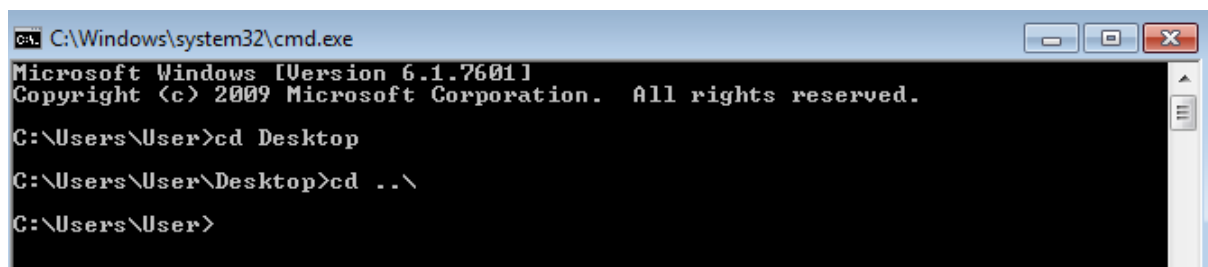
But let's not alter any file or folder on the Desktop. Instead, let's create a new folder. Do you recall the command to make a new folder? That's right, it's: **mkdir**.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\User>cd Desktop
C:\Users\User\Desktop>mkdir hyperion
```

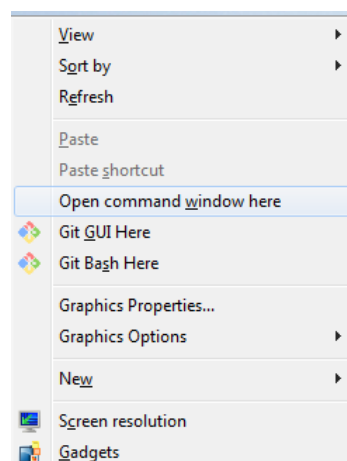
Notice that we have made a new folder on the Desktop called 'hyperion'. It's that simple! So, now that we have done what we wanted to do on our Desktop, how do we get back to where we were, i.e. how do we navigate **backwards**?



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\User>cd Desktop
C:\Users\User\Desktop>cd ..\
C:\Users\User>
```

To navigate two directories back, we would have to type, `cd ..\.`. However, navigating back and forth may seem tedious to do. Wouldn't it be nice if we could figure out a way in which we could open a Command Window in any directory with minimal effort? Fortunately, you can with Windows! Simply hold shift and right-click on a folder or empty space to open a Command Window in that directory:



Your computer may display “Open PowerShell window here”. PowerShell is similar to the command window and will accept most of the same commands.

## SCRIPT FILES

As you advance in your skills as a web developer, you may at times find that there are certain commands that you repeatedly use. Instead of retyping these commands into the command line repeatedly, you can create a script file that contains these sets of commands and that can be executed as needed. Often such files will be executed periodically, e.g. daily, weekly, monthly etc. In Windows, we can create **batch files** and in Mac and Linux systems, we create **shell scripts**.

### Batch files

A batch file is a script file in Windows. Batch files are normally used by individuals who run the same commands frequently. Instead of typing out the commands each time, the commands are simply placed in a batch file. To execute the commands contained within a batch file, you can simply double-click them.

The batch file consists of a series of commands to be executed by the command line. To create a batch file, you can open a plain text editor (e.g. Notepad) and navigate to File > Save As, and in the “Save As” window, input the name for your batch file and then add a “**.bat**” extension, e.g. *mybatch.bat*.

### Bash (Shell) scripts

To create a shell script:

1. Open a text editor (e.g. gedit or TextEdit).
2. Add the following instruction: **#!/bin/bash** to the first line of the script file.
3. On the following lines enter the instructions that you would usually type into the terminal, one instruction per line.
4. Save the file. It is not a requirement but it is common practice to save your file with a “**.sh**” extension. To save the file properly you may need to specify that the file is a plain text file. Do this by selecting Format > Make plain text.
5. Make this file executable by typing the following into the command line: **chmod +x myscript.sh** where *myscript.sh* is the name of the script file.
6. To run the script type: **sh myscript.sh** where *myscript.sh* is the name of the script file.

## SPOT CHECK

Let's see what you can remember from this section.

1. What are the two benefits of using the command line as a web developer?
2. What is the command to display the name of or change the current directory?
3. What is the command to display a list of files and subdirectories in a directory?

## PACKAGE MANAGERS

Package managers provide a means to install packages, organise them in your file system, and manage dependencies between packages. A dependency is when a package depends on another particular version of another package to function. If you have two different packages that rely on different versions of another package, this creates problems if you don't have a package manager. It can also be very time consuming to find and install all the dependencies for a package. Package managers also check for known vulnerabilities that may pose a security risk. You can use a package manager to share packages you have created with others.

There are two types of package managers: those for **operating systems** and those for **programming languages**. Package managers are linked to software repositories where the packages or software are stored.

Some common operating system package managers include:

- **Chocolatey** for Windows ([Chocolatey repository](#))
- **HomeBrew** for MacOS ([Homebrew repository](#))
- Linux – each distribution has its own package manager
  - **DNF** for Fedora
  - **APT** for Ubuntu

As mentioned, programming languages typically have language-specific package managers. Examples of some programming languages and commonly used package managers are summarised in the table below:

Language	Package manager	Software repository
JavaScript	npm	<a href="#">Node package manager</a>
JavaScript	yarn	<a href="#">Yarn registry</a>

**npm** is the most commonly used package manager for JavaScript and Node.js projects, providing a vast array of packages, libraries, and tools for front-end and back-end development. However, for projects that emphasise faster and more reliable package installation, **yarn** is a good alternative. Built on top of npm, yarn is known for its enhanced performance, parallel installations, and caching capabilities.

The package manager you use depends on your preferences and the project's specific needs. Some key considerations to keep in mind are:

- **Project requirements:** Ensure that the package manager you choose supports the necessary libraries and dependencies.
- **Community support:** Look for an active and supportive community to provide resources and troubleshooting assistance.
- **Performance and reliability:** Assess performance factors like speed and consistency vs comprehensive package registry management, especially for large complex projects.
- **Compatibility:** Consider compatibility to ensure that it integrates seamlessly with your tools and workflows.
- **Package versions:** Check whether the package manager is frequently updated to ensure your project benefits from bug fixes and new features.
- **Documentation:** Always review the documentation and ease of use for a smooth development process.



## SETTING UP YOUR DEVELOPMENT ENVIRONMENT

Please systematically and carefully follow the steps just below. Remember to read the following instructions in full before you implement them:

1. Visit <https://code.visualstudio.com/>.
2. Download the version of VS Code that matches your operating system (OS). Alternatively, you can follow the instructions stated at the following links for the corresponding operating system families:
  - a. Windows: <https://code.visualstudio.com/docs/setup/windows>
  - b. macOS: <https://code.visualstudio.com/docs/setup/mac>
  - c. Linux: <https://code.visualstudio.com/docs/setup/linux>
3. For all operating systems, ensure that your environment paths are up to date concerning your installation.
4. If you're concerned about opt-out telemetry with Visual Studio Code, please turn it off by using the instructions from:  
  
[https://code.visualstudio.com/docs/getstarted/telemetry#\\_disable-telemetry-reporting](https://code.visualstudio.com/docs/getstarted/telemetry#_disable-telemetry-reporting).
5. Next, you are going to need to download Node.js to be able to run your code in VS Code. To do this, go to <https://nodejs.org/en/download/> and download the correct version for your operating system.
6. Open the installer and follow the onscreen instructions.

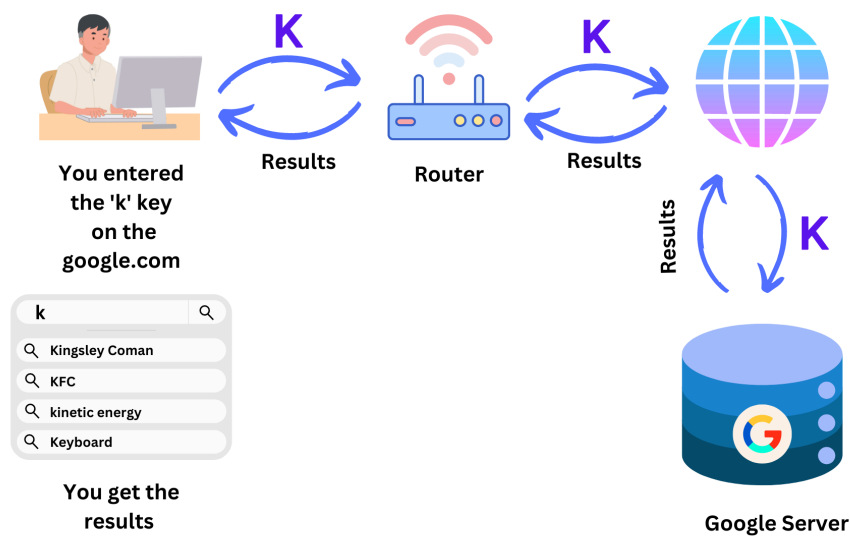
If you run into any trouble, submit a query via your dashboard for **assistance**.

Once you have set up your development environment, you are ready to start programming! You will use Visual Studio Code (VS Code) as your integrated development environment (IDE) to open all text files (.txt), HTML files (.html), CSS files (.css) and JavaScript files (.js).

You can visit the [JavaScript](#), [HTML](#), and [CSS](#) overview pages to learn how to use VS Code with these web development languages. If you've never programmed before, we strongly recommend that you watch the [introductory videos](#) and browse through the user guide to familiarise yourself with some [basic editing](#) and the [extension marketplace](#). If you encounter problems, [Stack Overflow](#) has several posts about VS Code.

## WHAT IS WEB DEVELOPMENT?

Every single keystroke you make on your computer sets off a mesmerising chain of events. It's an intricate dance of technology that begins at your fingertips and, with the magic of web development, could end up fetching data from a server halfway around the world, as shown in the “Google Query example” in the figure below. This captivating process underlies the vast potential of web development.



Our digital world functions due to the intricate dynamics of web development. It handles data transmission, processing, and display, powering every online interaction we make. Whether we are viewing the latest headlines, scrolling through our social media feeds, or managing our finances via online banking, web development plays a pivotal role. It's the foundation of the World Wide Web (www) – the vast global information system we depend on daily.

## THE WORLD-WIDE WEB

What is the World Wide Web? Well, it is a global information system consisting of web pages linked to each other using **hyperlinks**. These links allow us to navigate from one page on a website to another. They also allow us to navigate to pages from other websites from around the world. This linking technology creates the effect of an infinite web of information that we navigate daily.

Have you ever wondered who makes all these pages and ensures they're all linked together? The answer is web developers. They're a bit like the authors and

illustrators of a giant online book, creating and drawing each page and then connecting them so we can explore.



A note from the  
**HyperionDev Team**

Even though we can't imagine our lives without it, the World Wide Web is a rather recent invention. It was invented by Tim Berners-Lee, an English computer scientist, in 1989. Since its invention, it has expanded exponentially until it has become intricately interwoven into every part of our lives. Our work, entertainment, communication, and even our culture are strongly influenced by this powerful technology.

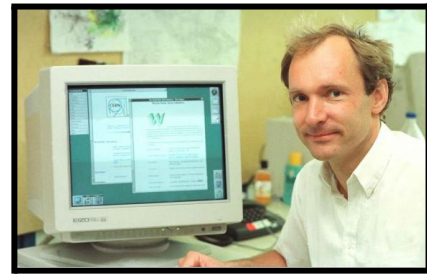


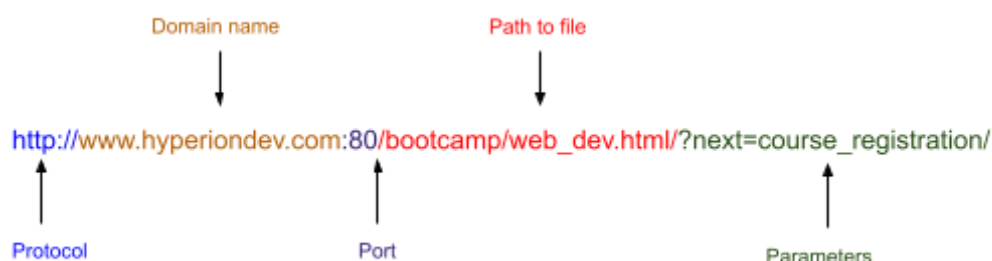
Figure 1: Tim Berners-Lee <sup>1</sup>

## CORE COMPONENTS OF THE WORLD WIDE WEB

The World Wide Web consists of three key components.

The first is called a **Universal Resource Identifier**, or URI. This is a unique identifier assigned to each page and resource on the web. It allows us to identify a specific page, video, audio, etc. file that we want over the internet. Now, a subset of URIs exists, known as **Uniform Resource Locators**, or URLs. These not only identify but also locate internet resources.

Consider the following fictional URL:



<sup>1</sup> "History of the Web – World Wide Web Foundation."  
<http://webfoundation.org/about/vision/history-of-the-web/>. Accessed 13 July. 2023.

As you can see, the URL contains a lot of information:

1. It identifies the protocol being used to send information. In the example above, the protocol being used is HTTP.
2. It identifies the domain name of the web server on which the resource can be found, e.g. `www.hyperiondev.com`.
3. It identifies the port on the server. In this example, the port number is given as port 80. In reality, if the default HTTP ports are used (port 80 is the default for HTTP, port 443 for HTTPS), they don't have to be given in the URL.
4. It gives the path to the resource on the web server, e.g. `/bootcamp/web_dev.html`.
5. Parameters can be passed using the URL. Parameters are passed as key-value pairs (`?key=value&key2=value2`), e.g. `?next=course_registration`

The second component is the language called **HyperText Markup Language (HTML)**, which is used to create web pages. Unlike other programming languages that allow us to create programs that perform tasks, this language is used to create the format of the page – what the contents are and how they are placed on the page.

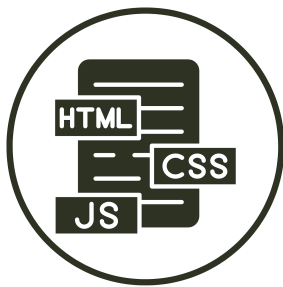
The third component is the protocol used to request and transfer web pages from one location to another. The internet is a frantically busy and complex medium of communication and, for us to ensure that we transfer things successfully from one location to another, we must have rules of transfer – a protocol – for devices to adhere to. This protocol is called **HyperText Transfer Protocol**, better known as **HTTP**.

## FRONT-END AND BACK-END WEB DEVELOPMENT

Within the realm of web development, two complementary engines drive our digital experiences: **front-end** and **back-end** development. Front-end development concentrates on crafting the user-facing elements and user experience, ensuring the visual aesthetics and interactivity align with user expectations. By contrast, back-end development focuses on behind-the-scenes functionality, facilitating data management, and handling server operations to deliver dynamic content to the user.

## Front-end development

Everything that you physically see, hear, and interact with on your screen when using the internet is considered to be the front end. For example, when you type a query into Google and hit enter, your request is whisked away to servers far removed from your immediate vicinity. A response is soon sent back to your computer, where your web browser interprets the data and renders it on your screen.



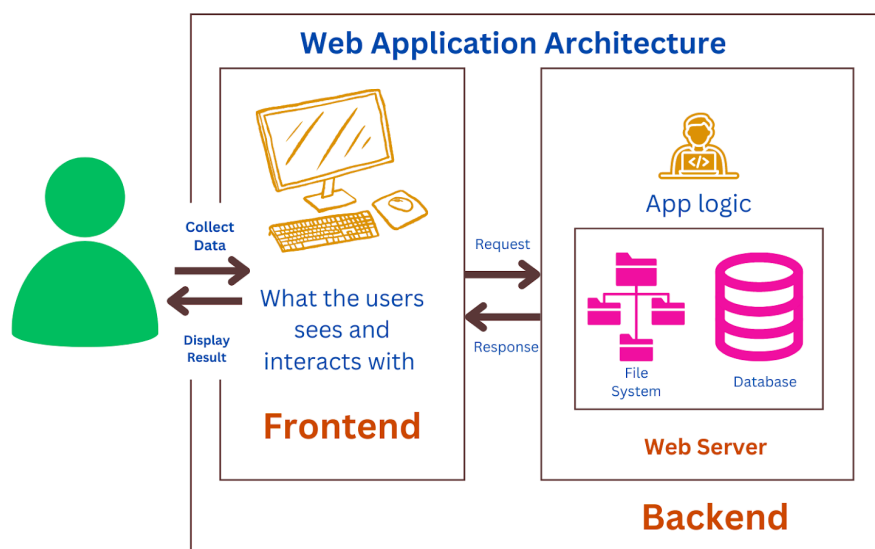
A trinity of powerful tools – [HTML](#), [CSS](#), and [JavaScript](#) – play an essential role in this process. HTML provides the foundational structure of a web page. CSS (Cascading Style Sheets) applies styling, adjusting the layout and adding colours to make the site visually appealing. Finally, JavaScript introduces interactivity, turning a static page into a dynamic experience.

## Back-end development

Back-end development refers to the processes and procedures occurring behind the scenes, on the server side. These unseen elements work together to ensure the seamless operation of a website or web application.

To illustrate, let's revisit our “Google query example”. When you type a query and press enter, the front-end part is the interface you see and interact with. However, the heavy lifting, including fetching the most relevant results according to your location, predicting your typing, and handling thousands of similar requests simultaneously, occurs in the back end.

Back-end developers primarily deal with servers, databases, and application logic. Applications are built using back-end languages such as JavaScript, Python, Java, PHP, and Ruby, to name a few. To manage, modify, and deliver data to the user interface, databases like MongoDB, MySQL, Oracle, PostgreSQL, and SQLite are utilised.



## FULL-STACK WEB DEVELOPMENT

A full-stack web developer should know how to build a website from the ground up, meaning that they should be able to create custom code to accommodate a client's unique needs and develop everything on the web page, from the site layout to features and functions.

Web development can be divided into three parts:

- **Client-side scripting:** This is code that executes in a web browser. It is what people see when they open a website.
- **Server-side scripting:** This is code that executes on a web server. It is everything behind the scenes that makes a website work.
- **Database technology:** This stores and manages all the data needed for a website.



A note from the  
**HyperionDev Team**

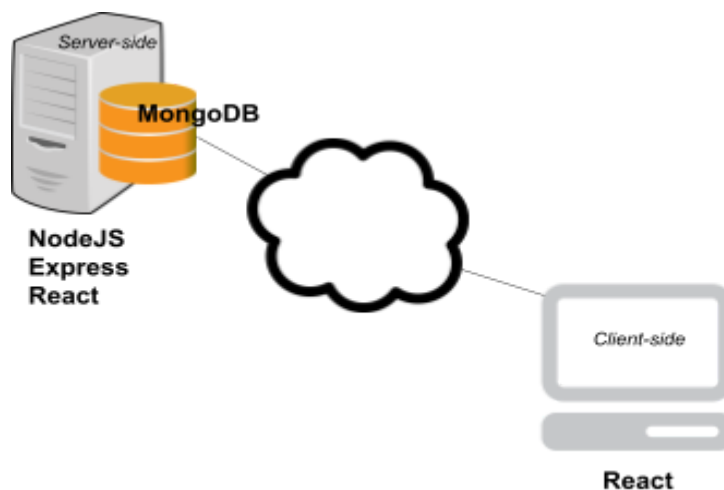
Check out this HyperionDev blog on a detailed comparison of **Front-End vs Back-End Web Dev.**

## THE MERN STACK

A **stack** is a term used to describe a **collection of technologies** that are used together to create a web application. There are several web development stacks including the LAMP stack, the MEAN stack, and the MERN stack.

One of the most popular approaches to full-stack web development using JavaScript is using the MERN stack. The MERN stack includes the following technologies:

- **MongoDB**: A non-relational database that stores data as documents or objects. It provides a JSON-based data-storage approach as an alternative to traditional SQL-based databases.
- **Express**: An un-opinionated web framework that simplifies server-side web development using Node.js. It streamlines the process of creating server-side applications with Node.js.
- **React**: A JavaScript library by Facebook for creating user views (components rendered in HTML). It offers a powerful toolset for building dynamic user interfaces on both the client and server side.
- **Node.js**: A runtime environment allowing JavaScript to be used not only in browsers but also on web servers. It enables server-side execution of JavaScript, expanding its capabilities beyond the browser environment.



### Extra resource

The MERN stack is a powerful stack in which to work. If you're able to build and deploy good MERN applications, it will greatly help your career prospects as a developer. Read more about the benefits of using the MERN stack [here](#).

## HTML AND SEMANTIC HTML

HTML is a markup language with which all front-end developers need to be comfortable. In this task, you will learn how to use HTML to create a basic static web page.

### INTRODUCTION TO HTML

HTML stands for Hypertext Markup Language. It is a language that we use to write files that tell the browser how to lay out the text and images on a page. We use HTML tags to define how the page must be structured.

#### HTML Tags

HTML tags are placed on the left and the right of the element you want to markup, to wrap around the element. For example:

```
<opening tag>Some text here.</closing tag>
```

This is the general pattern that we follow for all tags in HTML. There are a few exceptions, which we will discuss later. The words 'opening tag' and 'closing tag' are just placeholders we use to illustrate the pattern. Instead of those words, we are going to use special keywords, or elements, that modify the appearance of our web page.

Note that the opening and closing tags are not the same. The opening tag consists of an opening angled bracket (<), the name of the element, and a closing angled bracket (>). The closing tag consists of an opening angled bracket (<), a forward slash (/), then the name of the tag, and finally the closing angled bracket (>).

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first web page!</title>
  </head>
  <body>
    <p>I am learning to develop a dynamic web application.</p>
  </body>
</html>
```

*Example of HTML in a simple text file*



The HTML tags indicate to the browser what sort of structure the content is contained in. Note that HTML does not include the *style* of the content (e.g. font, colour, size, etc.), which is handled using CSS, but only the structure and content itself.

## HTML Elements

An element usually consists of an opening tag (`<element_name>`) and a closing tag (`</element_name>`), each containing the element's name surrounded by angled brackets, and the content in between these, as follows:

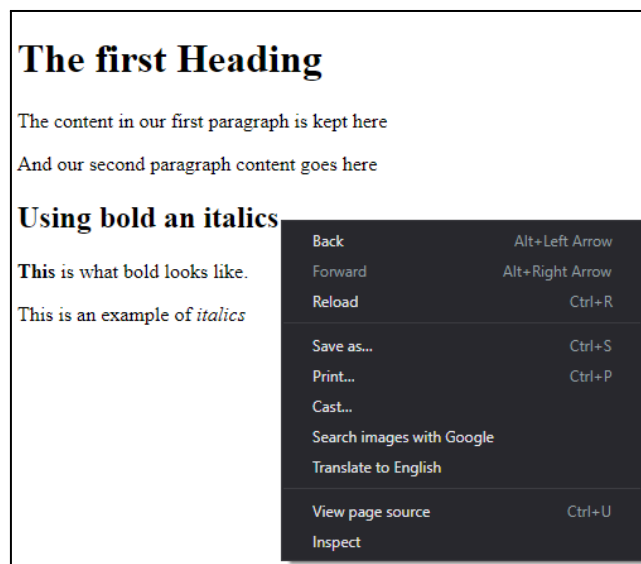
```
<element_name>...content...</element_name>
```

Example of the paragraph (`p`) HTML element:

```
<p>This element is going to result in this paragraph of text being displayed in the browser</p>
```

### Try this:

- Open the **example.html** file in your browser.
- Examine how the HTML page renders in the browser.
- Now, right-click in the browser and select the option 'View page source.'



- You will see the HTML used to create this web page, which includes many HTML tags. For example, you will notice the tags shown below:

```
<h2> Using bold an italics </h2>

<p> <b>This</b> is what bold looks like.</p> <!--This is also a tag and needs to be closed as shown here-->
<p> This is an example of <em>italics</em> </p> <!--em stands for emphasis, and thus makes it in italics -->
```

When the browser encounters the tag `<h2>` it knows to treat the information between the opening `<h2>` tag and the closing `</h2>` tag as a heading.

Similarly, the browser will display the information between the tags `<p>` and `</p>` as a paragraph of text.

## BASIC LAYOUT OF AN HTML PAGE

A general layout template that you can use, before even starting to worry about what sort of content you want to display, is set out below:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

A typical HTML document, as shown above, consists of the following elements:

- **doctype:** This indicates which version of HTML to load. The doctype is indicated at the top of the page and when typing 'html' it defaults to HTML5. This is one of the only elements that does not need a closing tag. Note that throughout HTML, capitalisation is very important.
- **html:** Next, we define what content is to follow within the `<html>` tags (note the closing tag at the bottom). Within this `<html>` element, we introduce two other elements, namely `<head>` and `<body>`. Notice that although each of the tags is located on a separate line, we still have opening tags matching their corresponding closing tags. Notice how the `<html>` tag wraps around its contents. We use a nested order to structure tags on our web page; this means that tags are contained within other tags, which may themselves contain more tags.
- **head:** This contains metadata about the page.
- **body:** This contains the actual content.

**Note:** It is important to understand how elements are nested because one of the most frequent mistakes that students make with HTML is getting the order all

mixed up. For example, it would be wrong to have a closing body tag (`</body>`) after a closing html tag (`</html>`) because the body element should be completely contained, or nested, within the `<html>` element. It should also be noted that white space is ignored by the browser, so you can lay out the physical spacing of the elements as you please.



## Code hack

### Hack to prevent unexpected errors:

In VS Code, open a blank HTML file and type an exclamation mark “!” at the beginning of the document. Then, hit “Enter” and you will instantly generate the basic HTML skeleton to kick-start your web development.

## ATTRIBUTES

Attributes describe the objects created by HTML elements. Attributes therefore provide extra information about HTML elements. All attributes are always written in the opening tag of an HTML element.

Now let’s look at page titles. Consider the following code:

```
<title id="myTitle">My first web page</title>
```

In this case, the element is of type **title**. Next, we have an **id**, which is an attribute of the element (**title**), and has the value “myTitle”. Attributes like this are used mainly for CSS and JavaScript (IDs are not compulsory, but they can come in very handy). Then there is a closing bracket `>` which indicates that you have finished defining the attributes of the element.

## COMMON HTML ELEMENTS

We have already encountered some of the common elements that are used to create most web pages. Some of these (and some new elements) are summarised below:

- A piece of metadata that should be included in all web pages is the `<title>` element. The `<title>` element:
  - defines a title in the browser tab,
  - provides a title for the page when it is added to favourites. and
  - displays a title for the page in search engine results.

As noted before, metadata should be contained in the `<head>` of the HTML document.

Example of a title element:

```
<head>
  <title>Portfolio</title>
</head>
```

- Use **headings** to show the structure of your web page, as you would when creating a document with a word processor, although in HTML they serve an additional function. Headings enable search engines to index the structure and content of your web pages. There are six heading elements you can use, `<h1>` to `<h6>`, where `<h1>` is used for the most important heading and `<h6>` for the least important.

Example of a heading element:

```
<h1>Online portfolio of work</h1>
<h2>About me</h2>
```

- Add paragraphs of text using the `<p>` element as follows:

```
<p>This is an example of a paragraph. Paragraphs usually contain more
text than headings and are not used by search engines to structure the
content of your web page. </p>
```

- **Line breaks:** To do the equivalent of pressing enter to get a line break between text, use the break (`<br>`) element. This element does not have a matching closing tag. This should make sense because there is no content that you could put within the `<br>` element. Elements like this, with no content or matching closing tags, are known as void elements.
- **Horizontal rule:** This is another void element. By adding the HTML element `<hr>` to your web page, you will create a horizontal rule.
- **Lists:** Lists can either be **ordered lists** `<ol>` or **unordered lists** `<ul>`. An ordered list is numbered, i.e. 1, 2, 3, etc., whereas an unordered list uses bullet points. In lists, keeping track of how deeply nested the various elements may be is **very** important. We highly recommend that you use indentations to keep track of which elements fall under which other elements. Remember

that indentation and “whitespace” do not affect the layout of the elements on the web page, they just help to make your code more human-readable.

- **Unordered Lists:** An unordered list element includes *list items* and thus has the tag `<li>`. To create an unordered list with three items in it, we could write the following:

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

Output example:

- Item 1
- Item 2
- Item 3

- **Ordered Lists:** Ordered lists work almost the same as unordered lists, except that you use the tag `<ol>`. You input list items in the same way as shown above. Instead of having bullet points, these list items are numbered.

```
<ol>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ol>
```

Output Example:

1. Item 1
2. Item 2
3. Item 3

- **Tables:** Tables work similarly to lists in terms of nesting elements. First, define the table element using the `<table>` tag. Then, you can add a table header `<th>` inside the table row tag. Below the headers, you can enter the data into the rows. Have a look at the example below:

```
<table>
  <tr>
    <th>Header 1</th>
    <th>Header 2</th>
    <th>Header 3</th>
  </tr>
  <tr>
```

```

    <td>Row 1, cell 1</td>
    <td>Row 1, cell 2</td>
    <td>Row 1, cell 3</td>
  </tr>
  <tr>
    <td>Row 2, cell 1</td>
    <td>Row 2, cell 2</td>
    <td>Row 2, cell 3</td>
  </tr>
  <tr>
    <td>Row 3, cell 1</td>
    <td>Row 3, cell 2</td>
    <td>Row 3, cell 3</td>
  </tr>
</table>

```

The table element is defined within the opening and closing tags. Immediately within these tags, there is a *table row* indicated by **<tr>** which also has a closing tag. Within that first table row, there is a **<td>** tag which indicates that there is *table data*. A table is shown in the **html\_example2.html** file provided to you. Have a look at the code and try to correlate which elements code for which visual effects on the web page. You can also try making small changes to the example file to see what effect this has when you refresh the displayed web page.

## LINKS

You can add links to your web page as follows:

```
<a href="example.com">link text</a>
```

The **<a>** element stands for “**a**nchor” and is used to add links on a web page. The **href** attribute stands for “**h**ypertext **r**eference”, i.e. to anchor a link using HTML. Using this element, you can link to other pages within your website, as well as to external web pages. You can also enable users to send an email by using the **mailto:** attribute within the opening anchor tag, as shown below. Note that the **mailto:** attribute takes an email address as a parameter, and the attribute and its parameter must all be enclosed in double quotes, like this:

```
<a href="mailto:example@hyperiondev.com">Contact us</a>
```

## Linking to other places on your web page

Often on your web page, you will want your users to be able to click on a link that will then take them to another part of the same page. Think about the “back to the top” button – you click on this and you are suddenly viewing the top of the page again!

To do this, we need to use *ID* attributes. An ID is used to identify one of your HTML elements, such as a paragraph, heading, or table. We can then use the link tag to make the text or image a link that the user clicks on to take them to whichever address we choose!

An ID can be assigned to any of your elements, as follows:

```
<h1 id="theHeading">My first web page</h1>
```

Notice how the attribute **id** is within the opening tag.

Now that we have this heading, we can look at how to reference it within our text. We use the `<a>` tag which shows which address we are using. To reference a structure with an ID, we need to precede the value assigned to the **id** attribute with a **#**, otherwise, the browser will think you are looking for a website.

```
<h1 id="theHeading">My first web page</h1>
<a href="#theHeading">Back to top</a>
```

Open the **links\_attributes\_images\_eg.html** example file. It contains the elements shown above. Notice how it makes the text “Back to top” look like a hyperlink (blue and underlined). When this is clicked, it will take you to the Heading with the **id** “theHeading”.

## Linking to other web pages

Similarly, we can link to another web page. This can be achieved as follows:

```
<a href = "http://www.hyperiondev.com">This cool place!</a>
```

The “**http://**” in front of the address lets the browser know that you are linking to an external website rather than a file on your system.

However, you aren’t limited to creating links through text! All the content that is between the `<a>` tags is what is to be clicked on to get to the destination address.

With the link specified above, if you click on the link it will change the window you’re currently on. However, what if you wanted to open the link’s destination address in a new tab? You can add an attribute to the link tag called **target** which

specifies how the link should be opened, e.g. in the same window, new browser instance, or new tab. Using **target="\_blank"** will open the link in a new tab instead of changing your current tab to the specified web address. This is very useful when you want to keep a user on your website and don't want them to leave every time they follow an embedded link.

To open in a new tab, simply modify the link as follows:

```
<a target="_blank" href="http://hyperiondev.com">This cool place!</a>
```

## IMAGES

We can add images to our website using the **<img>** element as shown below:

```
<!-- First example -->


<!-- Second Example -->

```

There are a few things to note about the **<img>** element.

- Unlike most of the other elements we have explored so far, the **<img>** element doesn't have a closing tag.
- The **<img>** element has several attributes that can describe it. These include:
  - **src=** This attribute gives the path to the location of the image, i.e. the *source* of the image.
  - **alt=** The **alt** attribute defines the *alternate text* that will be displayed if the image won't display. This is also useful to extend the accessibility of your web page, as page-reader tools for the visually impaired will be able to read the alternate text to users. As such, it is good practice to use the **alt** attribute to provide a succinct description of the image.
  - Intuitively, the **height** and **width** attributes define the height and width of the image.
- The **src** attribute can point to a URL or a file location. In the first example above, the first image uses a URL as the source of an image. The second example shows how the **src** attribute is defined to display an image named



`image1.jpg` that is stored in a folder named *images* that resides in the same folder as your web page.

A quick note on the format for relative file paths:

- If `image1.jpg` is in the same folder as the page we're adding it to, we simply write ``
- If `image1.jpg` is located in the `images` folder at the root of the web page, we write ``
- If `image1.jpg` is located in the folder one level up from the web page, we write `<img src = "../image1.jpg">`



### Take note:

When adding images to your web page, it is important to remember that the page may be viewed on many different devices with widely differing screen sizes, resolutions, etc. You want the images to look good independent of the device that is used to view the page. Thus, creating responsive images (images that work well on devices with widely differing screen sizes and resolutions) is important. To see how to create responsive images, see [here](#) and consult Chapter 15 of "HTML5 notes for professionals" (the additional reading provided with this task).

## FORMS

HTML forms allow users to submit information and interact with websites. We will start by exploring the key concepts of a simple form. While our form won't be functional at this stage, we'll focus on understanding the fundamental structure and elements of HTML forms.

```
<form>
  <label>First name:</label>
  <input type="text" />
  <label>Surname:</label>
  <input type="text" />
  <label>Gender:</label>
  <select>
    <option value="male">Male</option>
    <option value="female">Female</option>
    <option value="non-binary">Non-binary</option>
    <option value="other">Other</option>
  </select>
</form>
```

```
</select>
<label>Age:</label>
<input type="number" />
</form>
```

In the example above, we create a form to capture our user's biographical information. It can capture the following information:

- First Name
- Surname
- Gender
- Age

We expect the user to enter text for their name and surname. We, therefore, use the **input** element. This element has a **type** attribute with the **text** property assigned to it. This displays text boxes in the browser into which users can type input. We add labels to tell our visitors what information we want them to enter into the boxes.

The **select** element is used to create a drop-down menu from which users can select an option, instead of typing out their gender.

To see a list of other HTML input types, see [here](#) and consult Chapter 17 of "HTML5 notes for professionals" (the additional reading provided with this task).

## SEMANTIC HTML

The latest and most enhanced version of HTML is called Semantic HTML. Let's have a look at this and find out what Semantic HTML is and how to improve the way you create the markup for all your web pages.

One of the most important features of HTML5 is its semantics. The word semantic means "relating to meaning". HTML was originally designed to describe scientific documents semantically; it has since evolved to describe much more. Semantic HTML refers to writing HTML in a more comprehensible way, by better defining the different sections and layout of web pages.

When using Semantic HTML, we choose HTML elements based on their meaning, not on how they are presented. This approach also makes web pages more informative and adaptable, allowing browsers and search engines to interpret content better. Elements such as **<div>** and **<span>** are not semantic elements because they provide no context as to what is inside those tags.

## LIST OF COMMON SEMANTIC HTML ELEMENTS

**<header>** – The **<header>** element represents introductory content, typically a group of introductory or navigational aids.

The **<header>** element may contain:

- Heading elements
- A logo
- A search form
- An author name
- Navigational links

**<nav>** – The **<nav>** element represents a section of a web page the purpose of which is to provide navigation links, either within the current web page or to other web pages.

Example use cases of navigation sections:

- Menus
- Tables of contents
- Indexes

**<main>** - The **<main>** element represents the content of the **<body>** of a web page. The main content area consists of content that is directly related to, or expands upon, the central topic of a web page, or the central functionality of an application. Please look at the example (coming up) under the **<header>** element for an example of the **<main>** element.

**<section>** – This represents a generic standalone section of a web page, which doesn't have a more specific semantic element to represent it. A **<section>** should always have a heading, with very few exceptions.

**<figure>** – This represents self-contained content that is specified using the **<figcaption>** element (things like illustrations, diagrams, photos, and code listings). You can see an example of the **<figure>** element in the example below.

**<figcaption>** – This represents a caption or legend describing the rest of the contents of its parent **<figure>** element.

**<footer>** –This defines a footer for a web page or section. A **<footer>** typically contains information about the author of the section, copyright data, or links to related web pages.

Here is an example of the above semantic elements used to create a web page:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport"
      content="width=device-width, initial-scale=1 shrink-to-fit=no"/>
    <title>Home</title>
  </head>
  <body>
    <header>
      <a href="#">
        
      </a>
      <nav>
        <ul>
          <li><a href="#">Home</a></li>
          <li><a href="#cats">Cute Cat Express</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <section id="cats">
        <h1>Cute Cat Express</h1>
        <p>I love cats <em>so</em> much! Like, really, a lot.</p>
        <figure>
          
          <figcaption>A cat meme template</figcaption>
        </figure>
      </section>
    </main>
    <footer>Cat Memes &copy; 2024. All Rights Reserved</footer>
  </body>
</html>
```

## WHY USE SEMANTIC HTML?

### Accessibility

Screen readers and browsers can interpret Semantic HTML better, which makes web pages more accessible for people with disabilities.

## SEO

Using Semantic HTML can improve website search engine optimisation (SEO). SEO refers to the process of increasing the number of people that visit your web page. With better SEO, search engines are better able to identify the content of your website and weigh the most important content appropriately.

## Easy to understand

Writing Semantic HTML makes code easier to understand, making the source code more readable for other developers.

## READABILITY

As you start to create HTML pages with more elements, it becomes increasingly important to make sure that your HTML is easy to read. As you know, in web development, readability is an important principle! Code that is easy to read is easier to debug and maintain compared to code that is difficult to read.

Indenting your HTML is an important way of improving the readability of your code. For example, consider the HTML below:

```
<!DOCTYPE html><html><head>
<title>My first web page</title>
</head><body>
<form><label>First name:</label>
<input type="text"><br>
<label>Surname:</label>
<input type="text"><br>
<label>Gender:</label><br>
<select><option value="male">Male</option>
<option value="female">Female</option>
<option value="other">Other</option>
</select><br>
<label>Age:</label><br>
<input type="text"><br>
<input type="submit" value="Add user">
</form></body></html>
```

The above is perfectly correct HTML that will render properly in the browser. However, it is certainly not as easy to read and understand as the code below, which is properly indented:

```

<!DOCTYPE html>
<html>
  <head>
    <title>My first web page</title>
  </head>

  <!-- This is a comment. It is not interpreted as code and, thus
  will not affect your web page. Comments (along with indentation)
  can be used to improve the readability of your code. -->

  <body>
    <form>
      <label>First name:</label>
      <input type="text" />
      <label>Surname:</label>
      <input type="text" />
      <label>Gender:</label>
      <select>
        <option value="male">Male</option>
        <option value="female">Female</option>
        <option value="other">Other</option>
      </select>
      <label>Age:</label>
      <input type="number" />
      <input type="submit" value="Add user" />
    </form>
  </body>
</html>

```

As you can see in the example, the indentation is used to show which HTML elements are nested within which other HTML elements. All the elements on the web page are nested within the `<html>` element which provides the outer structure for the code.



## Code hack

Install the code formatter [\*\*Prettier\*\*](#) from the Extension tab in VS Code. This extension will automatically and consistently format your code each time you save the file.

Once this extension is installed, change the settings by typing “default” in the search box and selecting the “Text Editor” folder. Look for the option to set Prettier as the default formatter. Next, type “format” in the search box, navigate to “Text Editor -> Formatting” and check the box “Format on Save”.

## HTML SYNTAX

As a developer, you are going to learn many new languages. Each of these has its own rules which must be followed strictly for your instructions to be properly processed. As you know, the rules of a language are referred to as *syntax*. Examples of common HTML syntax errors include spelling the name of an element incorrectly, not closing tags properly, or closing tags in the wrong order. You are bound to make mistakes that will violate these rules and cause problems when you try to view web pages in the browser – we all make syntax errors! Often! Being able to identify and correct these errors becomes easier with time and is an extremely important skill to develop.

To help you identify HTML syntax errors, copy and paste the HTML you want to check into this helpful [\*\*tool\*\*](#).



## Extra resource

An **Extra Resource** for this task includes the e-book entitled **HTML5 Notes for Professional** by the ‘beautiful people of Stack Overflow’ which is included in this task's Dropbox folder.

## Instructions

Read and run the example files provided to you along with this task before doing the Practical Tasks. This will help you to become more comfortable with the concepts covered in the task before you start to apply them in the Practical Tasks.

## Practical Task 1

Follow these steps:

- Create a script file called **file\_cd**. (Remember to save this file with a .bat extension if you are using Windows)
  - Inside **file\_cd**, insert commands to create three new folders (directories). Name your folder as you wish.
  - Next, insert commands to navigate inside one of the folders you created and create three new folders inside this folder. Also, insert commands to remove two of the folders you created.

## Practical Task 2

Submit a pdf document entitled **WebFundamentals** in which you briefly answer the following questions:

1. What is the World Wide Web?
2. Explain the functional differences between a web application's front end and back end. See [here](#) for more information.
3. Describe what occurs on the back end during a web interaction using the "Google query example" (refer to page 10 of this PDF) or create your own example.
4. What is the MERN stack?



## Practical Task 3

In this task, you will create your own personal web page to showcase your HTML skills and serve as an online CV/résumé.

Although the web page's visual design is not the primary focus at this stage of your learning, pay attention to creating well-structured HTML code and presenting the content in a clear, readable and organised manner.

Follow these steps:

- In Visual Studio Code, create a new HTML file called **index.html**.
- Write HTML code to create a basic CV, including the following sections:
  - **Menu:** Include navigational links to direct users to different sections within your CV.
  - **Profile:** Display your name and a profile photo.
  - **Bio:** Write a short description of yourself, including your strengths, motivations, and aspirations.
  - **Skills:** Provide a list of both your soft and hard skills.
  - **Education:** List and describe your educational background.
  - **Work experience:** List and describe your roles and responsibilities. If you lack work experience, include any relevant volunteer work or completed projects instead. Provide links to the company's website and/or to your projects or portfolio, if applicable.
  - **Contact details:** Include your full name, contact number, email address, and links to your professional social media accounts.
- Ensure that your code includes the following mandatory elements:
  - **<nav>**: Navigation element for linking to different sections.
  - **<img>**: Image element to display your profile photo.
  - **<a>**: Anchor elements for linking to external websites or social media profiles.
  - **<ol>**, **<ul>**, and **<li>**: Ordered and unordered lists containing list elements for listing skills, education, and work experience.
  - **<hr>**: Horizontal rule element for visual separation.
  - **<h1>** to **<h6>** (any): Heading elements for proper content hierarchy.
- Feel free to customise the page and include additional elements as needed to suit your preferences, requirements, and creativity.

## Spot check answers

1. For web developers, it is helpful to use the command line when interacting with files, especially those created using libraries and frameworks, such as React. It also allows you to work with version control systems like Git.
2. `cd`
3. `ls` (MAC OS/UNIX) or `dir` (Windows).

## REFERENCES

Yang, Z. & Tamuri, A. (2015). *Getting Started with Mac OS X/ Linux Command Terminal*. Retrieved from <http://abacus.gene.ucl.ac.uk/software/CommandLine.MACosx.pdf>



Rate us

**Share your thoughts**

HyperionDev strives to provide internationally excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

[Click here](#) to share your thoughts anonymously.

