Hyperiondev

TASK

Getting Started With Your Bootcamp and HTML

Visit our website

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

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

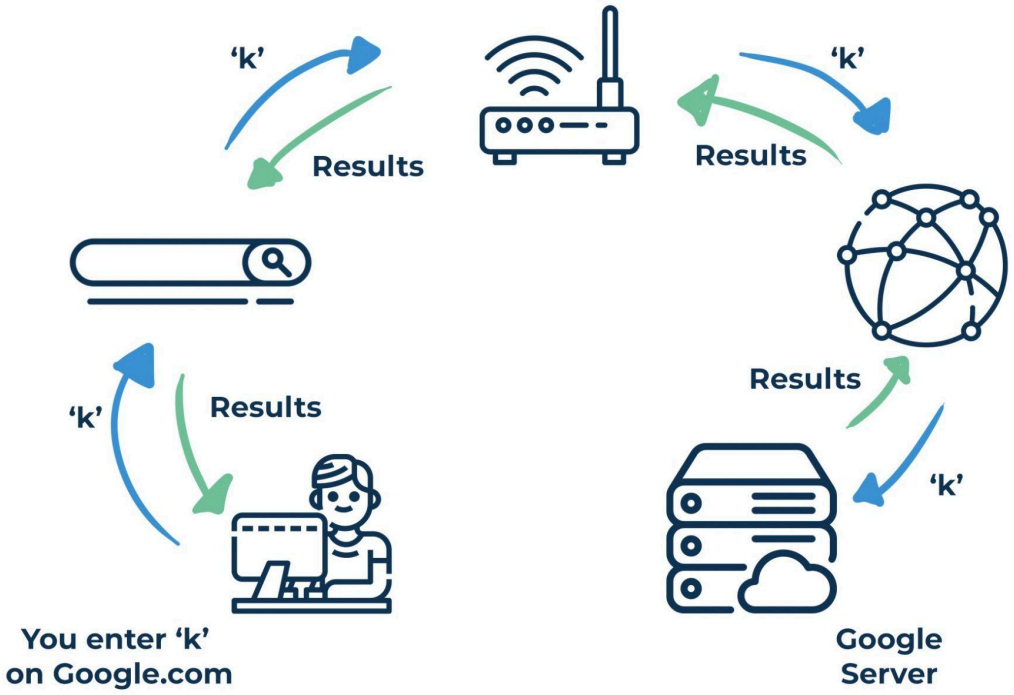
SET UP YOUR DEVELOPMENT ENVIRONMENT NOW

Before we get going, please consult the '**Additional Reading – Installation, Sharing, and Collaboration**' guide that accompanies this task. This reading will provide you with detailed instructions on setting up Visual Studio Code on macOS, Windows, and Linux. It includes steps for downloading, installing, setting up environment paths, and installing extensions or tools. It also offers alternatives for unsupported operating systems and solutions for common troubleshooting issues.

Please note: It is crucial to consult this reading and set up your development environment before beginning any practical coding components later on.

WHAT IS WEB DEVELOPMENT?

Every keystroke you make on your computer initiates a series of technical processes. For example, when you type a query into Google, your request travels through various systems and can retrieve data from a server anywhere in the world. This illustrates the core functionality and 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 is 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 are all linked together? The answer is web developers. They are 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 cannot 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.

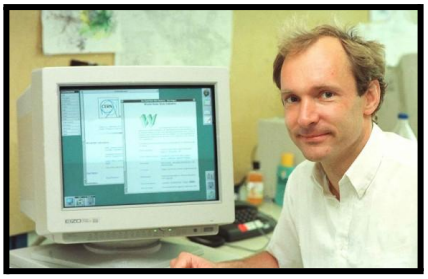


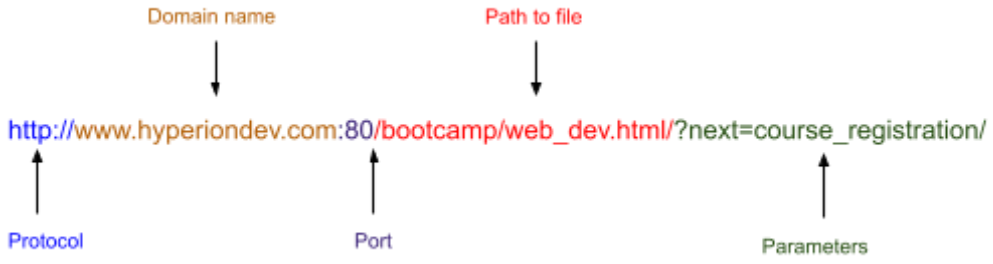
Figure 1: Tim Berners-Lee

CORE COMPONENTS OF THE WORLD WIDE WEB

The World Wide Web consists of three key components.

The first is called a **Uniform 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:



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 do not 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**.

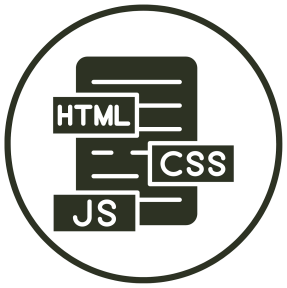
FRONT-END AND BACK-END WEB DEVELOPMENT

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

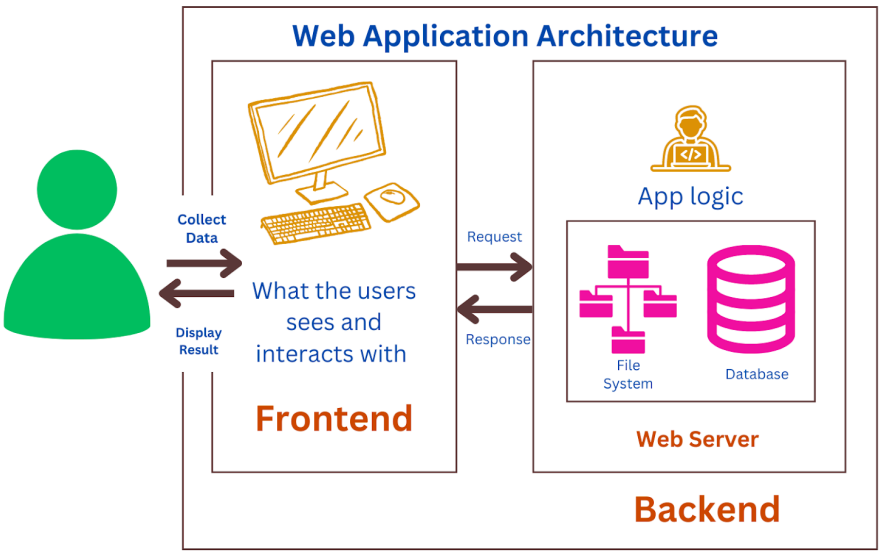
query into Google and hit enter, your request is whisked away to servers far removed from your immediate vicinity. A response is soon sent back to your computer, where your web browser interprets the data and renders it on your screen.



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 involves server-side processes that ensure a website or web application runs smoothly. Using the "Google query example," the front-end is what you interact with, but the back-end handles tasks like fetching relevant results, predicting typing, and managing multiple requests. Back-end developers work with servers, databases, and application logic using languages like JavaScript, Python, and PHP, and databases such as MongoDB and MySQL.



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**.

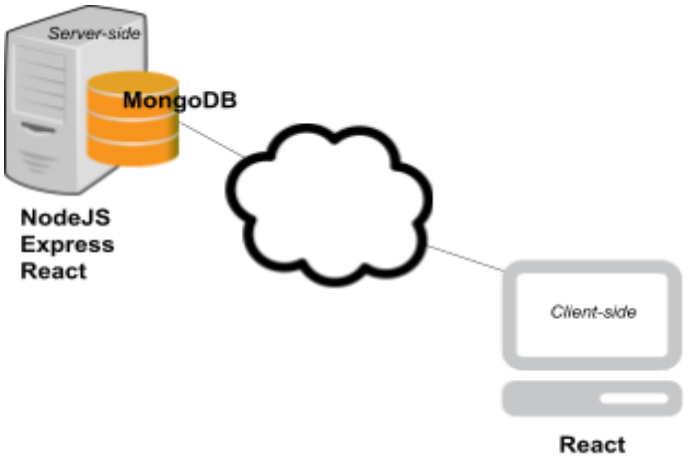
THE MERN STACK

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:

- **M**ongoDB: 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.
- **E**xpress: 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.

- **R**eact: 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.
- **N**ode.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 are 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**.

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 elements to define how the page must be structured.

HTML tags

HTML tags are placed on the left and the right of the content you want to markup, wrapping around it.

For example:

```
<opening_tag>Some text here.</closing_tag>
```

This is the general pattern that we follow for all elements 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 tag, 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, (`>`).

Look at the example code below:

```
<!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 (`<tag_name>`) and a closing tag (`</tag_name>`), each containing the tag's name surrounded by angled brackets, and the content in between these, as follows:

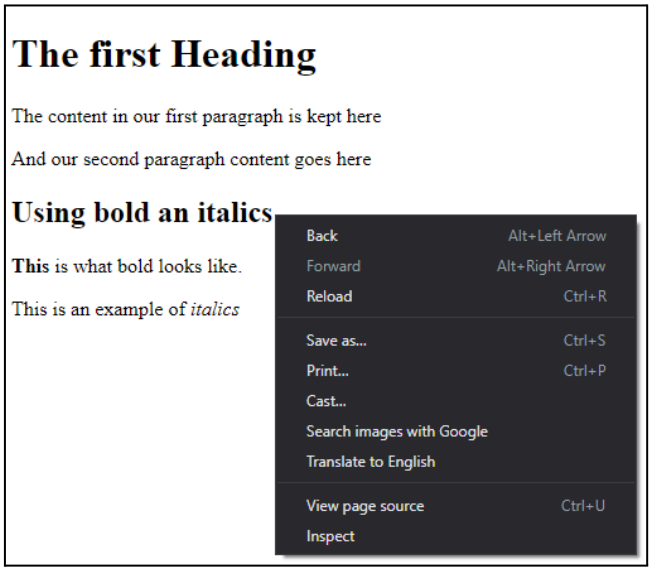
`<tag_name>...content...</tag_name>`

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 **html_example1.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:

```
<p>The content in our first paragraph is kept here</p>
<p>And our second paragraph content goes here</p>
<h2>Using bold and italics</h2>
```

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.

HTML Code Comments

Comments are used to explain and annotate your HTML code. They are not visible in the browser but can be seen in the source code. Use the following syntax for comments:

```
<!-- This is a comment -->
```

Example:

```
<!-- Main heading of the page -->
<h1>Welcome</h1>
```

In the above example, the comment indicates that the <h1> element serves as the main heading for the page. Comments help make your code clearer and easier to maintain.

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.
- **`html:`** Next, we define what content is to follow within the <html> tags (note the closing tag at the bottom). The **`<html>`** tag typically includes a **`lang`** attribute, which specifies the language of the document, helping search engines and browsers to understand the primary language used on the page.

  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. Metadata is information about the data on a page that helps define its content and structure.

- **`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="my-title">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 "my-title". 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 **`>`** that 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.

  Example of the line break element:

```
<p>
  First line of text.<br>
  Second line of text after a line break.<br>
  Third line of text with another line break.
</p>
```

- **Horizontal rule**: This is another void element. By adding the HTML element `<hr>` to your web page, you will create a horizontal rule.

  Example of the horizontal rule element.

```
<p>Content above the horizontal rule.</p>
<hr>
<p>Content below the horizontal rule.</p>
```

- **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 recommend using indentation to clearly show which elements are nested within others. Remember, indentation and spacing don't affect the web page's layout – they just make your code easier to read. Example:

```
<div>
  <h1>Title</h1>
  <p>Paragraph text here.</p>
</div>
```

  In this example just above, the `<h1>` and `<p>` tags are indented to show they are inside the `<div>` tag.

  - **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>` that 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 **ref**erence", 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:` scheme within the `href` attribute of the `<a>` element. For instance, `mailto:` is used within the `href` attribute to create a link that opens the user's default email application with a new email addressed to the specified email address. Both `mailto:` and the email address must be enclosed in double quotes.

```
<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 a "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="the-heading">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="the-heading">My first web page</h1>
<a href="#the-heading">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` "the-heading".

Linking to other web pages

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

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

The "`https://`" 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 are 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 do not 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="https://hyperiondev.com">This cool place!</a>
```

IMAGES

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

```
<img
  src="https://www.hyperiondev.com/static/images/couple.png"
  alt="Image of two people sitting cross-legged in front of a laptop."
```

```
/>
<img
  src="image1.jpg"
  alt="Image displaying a closing HTML tag with an orange background"
  height="200px"
  width="300px"
/>
```

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 are adding it to, we simply write `<img src="image1.jpg">`
- If `image1.jpg` is located in the images folder at the root of the web page, we write `<img src="images/image1.jpg">`
- 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.

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 will 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>
  <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**.

Now that we have covered basic HTML forms, let's explore how using Semantic HTML can enhance the structure and meaning of your web pages.

SEMANTIC HTML

Semantic HTML is the latest and most improved version of HTML, focusing on the meaning of elements rather than their presentation. It involves using specific HTML tags to define the structure and layout of web pages more clearly. This approach makes web pages more understandable and easier for browsers and search engines to interpret. Unlike non-semantic elements like `<div>` and `<span>`, semantic elements provide context for their content.

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="#">
        <img src="./images/logo.png" alt="my cat logo" />
      </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>
          <img src="./images/cat.jpg" alt="a cat meme template" />
```

```
          <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.

For help setting up Prettier after installation, refer to **this guide**.

HTML SYNTAX

As a developer, you will learn many new languages, each with strict syntax rules. Common HTML syntax errors include misspelling element names, improperly closing tags, or closing them in the wrong order. Everyone makes syntax errors, but with practice, identifying and fixing them becomes easier, making it a crucial skill to develop.

To help you identify HTML syntax errors, copy and paste the HTML you want to check into this helpful **tool**.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select “Request Review”, the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you’ve done that, feel free to progress to the next task.

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.

You are not likely to get it right the first time. Errors happen, and then they do, you need to know how to handle them. If you get stuck, read the **Defensive Programming** additional reading that accompanies this task.

Auto-graded task 1

Submit a pdf document or .txt file 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" or create your own example.
4. What is the MERN stack?

Be sure to place files for submission inside your task folder and click "Request review" on your dashboard.

Auto-graded task 2

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.

Be sure to place files for submission inside your task folder and click "Request review" on your dashboard.

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.