Topic 1.1: The HTML Boiler Plate

Starting Our HTML Learning

Because it is sometimes difficult to comprehend and remember theory without any like to real-wold application, rather than start with vocabulary definitions and discussion of rational of design, this first section will be more practical, jump in, and start creating a web page. Jargon will be introduced as needed, and once we have a feel for what web pages actually are and how they are made, we will then consider conventions and best practices for designing larger web projects.

Anatomy of an HTML Element

The framework of a web page is *Hypertext Markup Language*, more commonly referred to as HTML. An HTML file is made up of a single `DOCTYPE` declaration, then a number of HTML elements. An HTML element is either a single self-contained HTML tag, or an HTML opening tag, some optional element contents, and an HTML closing tag. Here is an example of each:

Self-contained HTML Tab	HTML Element with Opening and Closing Tags
`<hr>` self-contained HTML tag	`<h1>` ... `</h1>` opening tag closing tag

An HTML tag may be modified by *attributes*. These come after the tag name.

HTML Tab with attributes

attribute

`<h1 class="special">`Anatomy of an HTML Element`</h1>`

opening tag closing tag

Next, we will examine some simple HTML boiler plate code.

The HTML Boiler Plate

The term *boiler plate* comes from the industrial process of making rolled steel plates as a standard building block in the manufacture of steam boilers. In computer science, it refers to:

> ***boiler plate***: sections of code that are standard, repetitive, and have to be included in many places with little to no alteration.

Examine the following HTML file. The boiler plate is is **`bold font`**, while the page-specific contents of the page is in `gray`. HTML boiler plate can be inserted into an HTML file in VS-Code by typing an exclamation point, followed by the `<enter>` key on the keyboard.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport"
6            content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
```

```
</head>
<body>
    <h1>Web Page Heading</h1>
    <p>This is the first paragraph.</p>
    <p>This is the second paragraph.</p>
    <p>
        <a href="https://www.nielsenedu.com/index.html">
            A link to another web page
        </a>
    </p>
</body>
</html>
```

`<!DOCTYPE html>`

Like most technology does, HTML has gone through a number of versions. The current version of HTML is HTML5, and the HTML5 specification states that every HTML5 file should have as the first line `<!DOCTYPE html>`.

Previous versions of HTML required a complicated Document Type Declaration (`DOCTYPE`) that pointed to a specific HTML version's Document Type Definition (DTD), which is a formal rulebook that tells the browser about how to interpret the HTML file. An example of and HTML4 `DOCTYPE` declaration is given here:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
```

`<html lang="en"> … </html>`

The `<html>` element is the root element that wraps all other HTML content on a page. There must be only one `<html>` element in each HTML document.

The only part of your HTML file that exists outside the `<html>` element is the very first line: the `<!DOCTYPE html>` declaration.

Including the `lang` attribute use on the `<html>` tag is strongly advised as it is considered a fundamental best practice for accessibility and technical correctness.

- Screen readers use it to select the correct voice and pronunciation.
- Search engines can better index and serve the page to the correct language audience.
- Translation tools can use it to determine the language to translate.
- Browsers can apply language-specific typography rules, such as correct quotation marks or hyphenation.

The `<html>` element should only have two direct children: `<head>` and `<body>` elements. Any other element placed inside the `<html>` element will be automatically moved by the browser into the `<body>`.

`<head> … </head>`

The <head> element holds essential information about the page that is not the visible content itself, but is still crucial for the browser, search engines, and other services.

Attributes are rarely applied to the <head> element because the contents of this element are not displayed.

The contents of the <head> element will be expanded on in a later section.

<body> … </body>

The <body> element represents the visible content of an HTML document. Everything users see and interact with in the browser window, such as text, images, links, forms, and multimedia must be inside the <body> element.

Attributes are rarely applied to the <body> tag; perhaps an `id` or `class` attribute for styling.

Traditionally, some event handlers were included in the <body> tag, such as `onclick` or `onload`. This is discouraged in modern HTML because, for separation of concerns, functionality should be implemented in the JavaScript, not the HTML.

Inside the `<head>` Tag

`<title> … </title>`

A valid HTML file is required to have a `<title>` element within the `<head>` element (if not present, browsers will usually generate one for you). The contents of the `<title>` element is shown in the browser tab and search engine results.

`<meta charset="utf-8">`

Declares the character encoding for the document (e.g., `UTF-8`).

Over 90% of web pages use `UTF-8` encoding. Most modern text editors default to `UTF-8` encoding when saving a file. Just FYI, other possible encoding schemes include `UTF-16`, `ISO-8859-1`, `Shift-JIS` (Japanese), and `GBK` (Chinese). Except for perhaps some niche or legacy pages, all modern web pages should use `UTF-8` encoding.

`<meta name="viewport" ...>`

This tag controls the viewport's size and scaling for responsive design. Without it, a mobile browser may render the page as if it were on a wide desktop screen, resulting in a poor, zoomed-out user experience. The standard, recommended tag that provides a good mobile-friendly baseline without restricting user control follows:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

The "`width=device-width`" sets the viewport width to match the device's screen width, while "`initial-scale=1.0`" sets the initial zoom level when the page loads to `1.0` (100%), preventing the page from loading zoomed in or out.

Modern web browsers do not default to these setting to maintain backwards compatibility with older sites that were designed only for desktop. This tag basically tells the web browser: yes, the web site has been designed to be responsive to mobile.

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

The standard way to link an external CSS file (external CSS). This is the preferred method of adding CSS and we will be discuss this when we study CSS.

`<style> … </style>`

For embedding CSS directly in the HTML document (internal CSS). This should generally not be included in production web design. However, it is much better than using inline CSS, and is useful for quick prototyping or situations where you just need a quick solution, such as when writing an exam about web design.

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

For linking an external JavaScript file (external JavaScript). When placed in `<head>`, it often uses the `defer` attribute to delay the execution until after the page has been fully parsed and the DOM tree built.

Without the `defer` attribute, the page loading is paused while the script executes, and if the script refers to anything in the DOM (say using `selectElementById` or `querySelector`), it will not be found because the DOM tree has not been built. Thus, the `defer` attribute is usually included. The script will be loaded asynchronously when the `<script … defer>` is encountered.

`<script> … </script>`

For embedding JavaScript directly in the HTML document (internal JavaScript). This should generally not be included in production web design. However, it is considered much better than using inline JavaScript, and is useful for quick prototyping or situations where you need a quick solution, such as when writing an exam about web design. Since code cannot be deferred, `<script>` tags are generally placed near the end of the HTML file, just before the closing `</body>` tag so that the DOM tree will be built and the JavaScript can access the necessary portions of the DOM.

`<meta name="description" content="This web site...">`

Search engines like Google often display this text as the snippet below your page's title in search results. A good description can improve your click-through rate, so write a compelling, accurate summary that includes primary keywords naturally. Aim for about 150 to 160 characters, as search engines may truncate longer descriptions. Each page on your site should have a unique description relevant to its specific content.

`<meta http-equiv="refresh" content="5">`

This is an old, obsolete method to either:

- Refresh the current page after a set number of seconds.
- Redirect the user to a different URL after a delay.

The number of seconds is given in the `content` attribute. To redirect to another, add a url to the content attribute, as shown in this example:

```
<meta http-equiv="refresh" content="3; url=https://example.com/new-page">
```

The proper way to redirect to another page is for the server to send a HTTP return code that is in the `300`'s, which will instruct the browser to redirect. Search engines may penalize or distrust pages that use client-side (HTML) redirects. This redirect method has been included here because it has been presented in past papers of Pearson Edexcel.