web222

WEB222 - Week 5

Suggested Readings

- HTML: HyperText Markup Language on MDN
- HTML Basics
- Learning HTML: Guides and Tutorials
- HTML Reference

Running a Development Web Environment

Developing for the web requires at least 3 things pieces of software:

1. a proper code editor which, is aware of HTML, JavaScript, and CSS
2. a web client (i.e., browser), with developer and debugging tools
3. a web server

Code Editor

For our code editor, we will be using Visual Studio Code, which is a free (open source) code editor created and maintained by Microsoft. It also works on Windows, macOS, and Linux. Make sure you have downloaded and installed it on all the computers you will use for web development.

Web Client

For our web client we will use the many web browsers we introduced in Week 1, namely:

- Google Chrome for desktop and Android
- Microsoft Edge and Internet Explorer (IE)
- Apple Safari and Safari for iOS
- Mozilla Firefox
- Opera

There are many more, and you are highly encouraged to install as many as possible.

Web Server

We will also need a **web server** to host our web pages and applications. Installing and running a web server can be complicated. Industry grade web servers like Apache and nginx are free and can be installed and run on your local computer; however, they are much more complicated and powerful than anything we will need for hosting our initial web pages.

For our purposes, we will use a simple node.js based http servers. In order to use them, do the following:

1. Make sure you have installed node.js on your computer.
2. In a terminal window, navigate to the directory that you want your web server to host. For example `cd my-website`
3. Now start the web server by running the following command: `npx lite-server` .

This will download and run the necessary software, and show you a message like the following:

```
Did not detect a `bs-config.json` or `bs-config.js` override file. Using lite-server defau
** browser-sync config **
{ injectChanges: false,
  files: [ './**/*.{html,htm,css,js}' ],
  watchOptions: { ignored: 'node_modules' },
  server: { baseDir: './', middleware: [ [Function], [Function] ] } }
[Browsersync] Access URLs:
 --------------------------------------
       Local: http://localhost:3000
    External: http://192.168.1.110:3000
 --------------------------------------
          UI: http://localhost:3001
 UI External: http://localhost:3001
 --------------------------------------
[Browsersync] Serving files from: ./
[Browsersync] Watching files...
```

You can now open your web browser to `http://127.0.0.1:3000` and load your files. This uses the `http` protocol, and connects you to the special IP address `127.0.0.1` , also known as localhost (i.e., you can also use `http://localhost:3000`). The localhost IP address always refers to *this* computer, and allows you to connect network clients to your own machine. The final `:3000` portion of the URL is a port number. Together, `http://127.0.0.1:3000` means *connect using HTTP to my local computer on port 3000.*

NOTE: the second External IP address will be different than the above, but 127.0.0.1 will always be correct.

When you are done testing your web site, stop the web server by pressing `CTRL-C` in your terminal window. To run the server again, use `npx lite-server` .

NOTE: a previous version of this document recommended using the command `npx http-server`. This works well on macOS and Linux, but recently started to fail on Windows, see this bug.

HTML

HTML is the HyperText Markup Language. It allows us to write *content* in a document, just as we would in a file created by a word processor. Unlike a regular text file, it also includes structural and layout information about this content. We literally *mark up* the text of our document with extra information.

When talking about HTML's markup, we'll often refer to the following terms:

- tag: separated from regular content, tags are special text (names) wrapped in `<` and `>` characters, for example the image tag `<img>`.
- element: everything from an opening tag to the closing tag, for example: `<h1>Chapter 1</h1>`. Here an element is made up of an `<h1>` tag (i.e., opening Heading 1 tag), the text content `Chapter 1`, and a closing `</h1>` tag. These three together create an `h1` element in the document.
- attribute: optional characteristics of an element defined using the style `name` or `name="value"`, for example `<p id="error-message" hidden>There was an error downloading the file</p>`. Here two attributes are included with the `p` element: an `id` with value `"error-message"` (in quotes), and the `hidden` attribute (note: not all attributes need to have a value). Full list of common attributes.
- entity: special text that should not be confused for HTML markup. Entities begin with `&` and end with `;`. For example, if you need to use the `<` character in your document, you need to use `<` instead, since `<` would be interpreted as part of an HTML tag. ` ` is a single whitespace and `&` is the `&` symbol. Full list of named entities.

HTML Document

First HTML page ever created was built by Tim Berners-Lee on August 6, 1991.

Since then, the web has gone through many versions:

- HTML - created in 1990 and standardized in 1997 as HTML 4
- xHTML - a rewrite of HTML using XML in 2000
- HTML5 - the current standard.

Basic HTML5 Document

Here's a basic HTML5 web page:

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>My Web Page</title>
    </head>

    <body>
        <!-- This is a comment -->
        <h1>Hello World!</h1>
    </body>
</html>
```

Let's break this down and look at what's happening.

1. `<!doctype html>` tells the browser what kind of document this is, and how to interpret/render it
2. `<html>` the root element of our document: all other elements will be included with `<html>...</html>`.
3. `<head>` provides various information *about* the document as opposed to providing its content. This is metadata that describes the document.
4. `<meta>` an example of some piece of metadata, in this case defining the character set used in the document: utf-8
5. `<title>` an example of a specific (named) metadata element: the document's title, shown in the browser's title bar. There are a number of specific named metadata elements like this.
6. `<body>` the content of the document is contained within `<body>...</body>`.
7. `<!-- ... -->` a comment, similar to using `/* ... */` in C or JavaScript
8. `<h1>` a heading element (there are headings 1 through 6), which is a title or sub-title in a document.

Now let's try creating and loading this file in our browser:

1. Make a directory on your computer called `my-website`
2. Create a new file in `my-website` named `index.html`
3. Use Visual Studio Code to open your `my-website/index.html` file
4. Copy the HTML we just discussed above, and paste it into your editor
5. Save your `index.html` file
6. In a terminal, navigate to your `my-website` directory
7. Start a web server by typing `npx http-server`
8. Open your web browser (Chrome, Firefox, etc) and enter `http://localhost:8080` in the URL bar
9. Make sure you can see a new page with `Hello World!` in black text.

Now let's make a change to our document:

1. Go back to your editor and change the `index.html` file so that instead of `Hello World!` you have `This is my web page.`
2. Save your `index.html` file.
3. Go back to your browser and hit the **Refresh** button.
4. Make sure your web page now says `This is my web page.`

Every time we update anything in our web page, we have to refresh the web page in our browser. The web server will serve the most recent version of the file on disk when it is requested.

Common HTML Elements

There are dozens of HTML elements you'll learn and use, but the following is a good set to get you started.

Metadata

Information *about* the document vs. the document's content goes in various metadata elements:

- `<link>` - links from this document to external resources, such as CSS stylesheets
- `<meta>` - metadata that can't be included via other elements
- `<title>` - the document's title

Content Sections

These are organizational blocks within the document, helping give structure to the content and provide clues to browsers, screen readers, and other software about how to present the content:

- `<header>` - introductory material at the top of a document
- `<nav>` - content related to navigation (a menu, index, links, etc)
- `<main>` - the main content of the document. For example, a news article's paragraphs vs. ads, links, navigation buttons, etc.
- `<h1>, <h2>, ..., <h6>` - (sub) headers for different sections of content
- `<footer>` - end material (author, copyright, links)

Text Content

We organize content into "boxes," some of which have unique layout characteristics.

- `<div>` - a generic container we use to attach CSS styles to a particular area of content
- `<ol>` - an ordered list (1, 2, 3) of list items
- `<ul>` - an unordered list (bullets) of list items

- `<li>` - a list item in an `<ul>` or `<ol>`
- `<p>` - a paragraph
- `<blockquote>` - an extended quotation

Inline Text

We also use elements within larger text content to indicate that certain words or phrases are to be shown differently:

- `<a>` - an "anchor" element, which will produce a hyperlink, allowing users to click and navigate to some other document.
- `<code>` - formats the text as computer code vs. regular text.
- `<em>` - adds emphasis to the text (often in italics)
- `<span>` - another generic container, used to define CSS styles

Multimedia

In addition to text, HTML5 also defines a number of rich media elements:

- `<img>` - an element used to embed images in a document.
- `<audio>` - an element used to embed sound in a document.
- `<video>` - an element used to embed video in a document
- `<canvas>` - a graphical area (rectangle) used to draw with either 2D or 3D using JavaScript.

Scripting

We create dynamic web content and applications through the use of scripting:

- `<script>` - used to embed executable code in a document, typically JavaScript.

Examples:

- Lists: ordered and unordered
- Anchors: creating hyperlinks
- Images: using img
- Text: text sections
