

Lecture 8

- [Welcome!](#)
- [Routers](#)
- [DNS](#)
- [HTTP](#)
- [HTML](#)
- [CSS](#)
- [Frameworks](#)
- [JavaScript](#)
- [Summing Up](#)

Welcome!

- In previous weeks, we introduced you to Python, a high-level programming language that utilized the same building blocks we learned in C. Today, we will extend those building blocks further in HTML, CSS, and JavaScript.
- The internet is a technology that we all use.
- Using our skills from previous weeks, we can build our own web pages and applications.
- The *ARPANET* connected the first points on the internet to one another.
- Dots between two points could be considered *routers*.

Routers

- To route data from one place to another, we need to make *routing decisions*. That is, someone needs to program how data is transferred from point A to point B.
- You can imagine how data could take multiple paths from point A and point B, such that when a router is congested, data can flow through another path.
- *TCP/IP* are two protocols that allow computers to transfer data between them over the internet.
- *IP* or *internet protocol* is a way by which computers can identify one another across the internet. Every computer has a unique address in the world. Addresses are in this form:

`#.#.#.#`

- Numbers range from `0` to `255`. IP addresses are 32-bits, meaning that these addresses could accommodate over 4 billion addresses. Newer versions of IP addresses can accommodate far more computers!
- In the real world, servers do a lot of work for us.

- *TCP*, or transmission control protocol, is used to distinguish web services from one another. For example, `80` is used to denote HTTP and `443` is used to denote HTTPS. These numbers are *port numbers*.
- When information is sent from one location to another, an IP address and TCP port number are sent.
- These protocols are also used to fragment large files into multiple parts called *packets*. For example, a large photo of a cat can be sent in multiple packets. When a packet is lost, TCP/IP can request missing packets again from the origin server.
- TCP will acknowledge when all the data has been transmitted and received.

DNS

- It would be very tedious if you needed to remember an address number to visit a website.
- *DNS*, or *domain name systems*, is a collection of servers on the internet that are used to route website addresses like *harvard.edu* to a specific IP address.
- DNS simply hold a table or database that links specific, fully qualified domain names to specific IP addresses.

HTTP

- *HTTP* or *hypertext transfer protocol* is an application-level protocol that developers use to build powerful and useful things.
- When you see an address such as `https://www.example.com` you are actually implicitly visiting that address with a `/` at the end of it.
- The *path* is what exists after that slash. For example, `https://www.example.com/folder/file.html` visits `example.com` and browses to the `folder` folder and then visits the file named `file.html`.
- `https` in this address is the protocol that is used to connect to that web address. By protocol, we mean that HTTP utilizes `GET` or `POST` *requests* to ask for information from a server. For example, you can launch Google Chrome, right-click, and click `inspect`. When you open the `developer tools` and visit `Network`, selecting `Preserve log`, you will see `Request Headers`. You'll see mentions of `GET`. This is possible in other browsers as well, using slightly different methods.
- Generally, after making a request a server, you will receive the following in `Response Headers`:

```
HTTP/1.1 200 OK
Content-Type: text/html
```

- This approach to inspecting these logs may be a bit more complicated than need be. You can analyze the work of HTTP protocols at cs50.dev (<https://cs50.dev>). For example, type

the following in your terminal window:

```
curl -I https://www.harvard.edu
```

Notice that the output of this command returns all the header values of the responses of the server.

- Similarly, execute the following in your terminal window:

```
curl -I http://www.harvard.edu
```

Notice that the `s` in `https` has been removed. The server response will show that the response is `301` instead of `100`, meaning that the website has permanently moved.

- Further, execute the following command in your terminal window:

```
curl -I https://harvard.edu
```

Notice that you will see the same `301` response, providing a hint to a browser of where it can find the correct website.

- Similar to `301`, a code of `404` means that a specified URL has not been found. There are numerous other response codes, such as:

```
200 OK
301 Moved Permanently
302 Found
304 Not Modified
304 Temporary Redirect
401 Unauthorized
403 Forbidden
404 Not Found
418 I'm a Teapot
500 Internal Server Error
503 Service Unavailable
```

- It's worth mentioning that `500` errors are always your fault as the developer. This will be especially important for next week's pset, and potentially for your final project!
- We can send more complicated commands to the server. For example, we can attempt the following:

```
GET /search?q=cats HTTP/1.1
Host: www.google.com
```

Notice that not only are we specifying a path but also user input using the `?` mark. `q` is used to denote *query*, passing `cats` to it.

- If you manually type `google.com/search?q=cats` into your web browser address bar, it will manually query Google for results related to `cats`.

HTML

- *HTML* or *hypertext markup language* is made up of *tags*, each of which may have some *attributes* that describe it.
- In your terminal, type `code hello.html` and write code as follows:

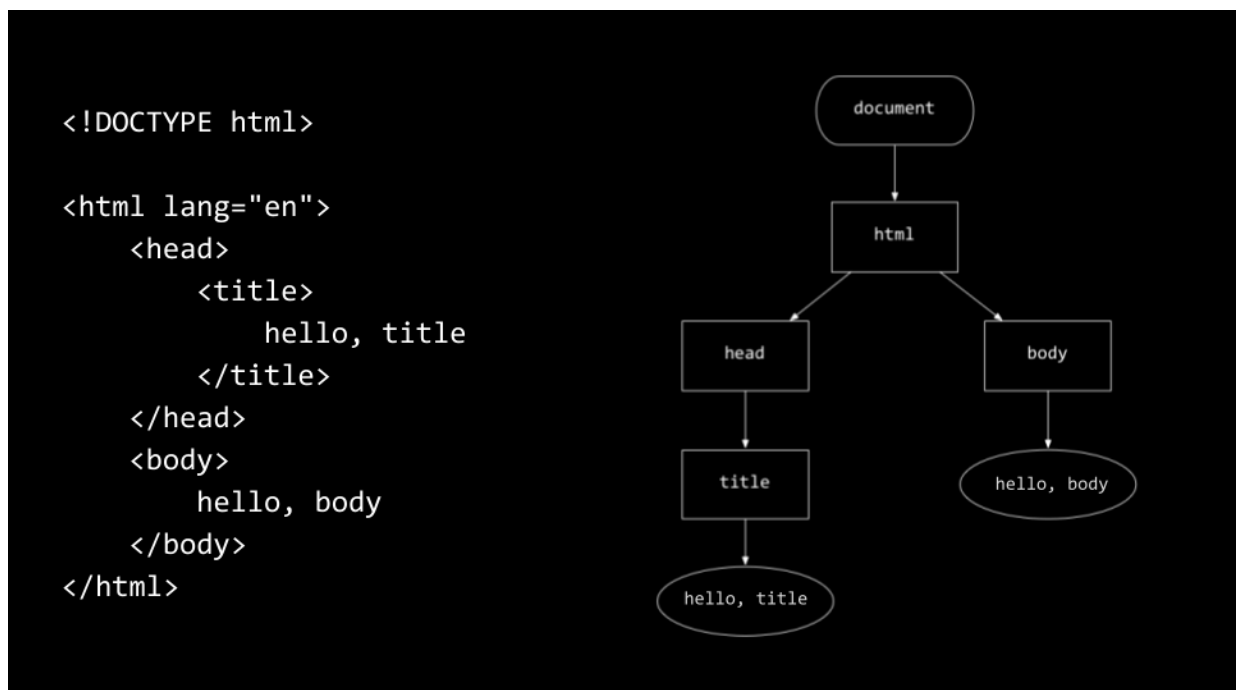
```
<!DOCTYPE html>

<!-- Demonstrates HTML -->

<html lang="en">
  <head>
    <title>hello, title</title>
  </head>
  <body>
    hello, body
  </body>
</html>
```

Notice that the `html` tag both opens and closes this file. Further, notice the `lang` attribute, which modifies the behavior of the `html` tag. Also, notice that there are both `head` tags and `body` tags. Indentation is not required but does suggest a hierarchy.

- You can serve your code by typing `http-server`. This serve is now available on a very long URL. If you click it, you can visit the website with your own code.
- When you visit this URL, notice that the file name `hello.html` appears at the end of this URL.
- The hierarchy of tags can be represented as follows:



- The browser will read your HTML file top to bottom and left to right.
- Because whitespace is effectively ignored in HTML, you will need to use `<p>` paragraph tags to open and close a paragraph. Consider the following:

```
<!DOCTYPE html>

<!-- Demonstrates paragraphs -->
```

```

<html lang="en">
  <head>
    <title>paragraphs</title>
  </head>
  <body>
    <p>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus c
    </p>
    <p>
      Mauris ut dui in eros semper hendrerit. Morbi vel elit mi. Sed sit
    </p>
    <p>
      Aenean venenatis convallis ante a rhoncus. Nullam in metus vel dia
    </p>
    <p>
      Integer at justo lacinia libero blandit aliquam ut ut dui. Quisque
    </p>
    <p>
      Suspendisse rutrum vestibulum odio, sed venenatis purus condimentu
    </p>
    <p>
      Sed quis malesuada mi. Nam id purus quis augue sagittis pharetra.
    </p>
  </body>
</html>

```

Notice that paragraphs start with a `<p>` tag and end with a `</p>` tag.

- HTML allows for the representation of headings:

```

<!DOCTYPE html>

<!-- Demonstrates headings (for chapters, sections, subsections, etc.) -->

<html lang="en">
  <head>
    <title>headings</title>
  </head>
  <body>
    <h1>One</h1>
    <p>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus c
    </p>
    <h2>Two</h2>
    <p>
      Mauris ut dui in eros semper hendrerit. Morbi vel elit mi. Sed sit
    </p>
    <h3>Three</h3>
    <p>
      Aenean venenatis convallis ante a rhoncus. Nullam in metus vel dia
    </p>

```

```

    <h4>Four</h4>
    <p>
        Integer at justo lacinia libero blandit aliquam ut ut dui. Quisque
    </p>

    <h5>Five</h5>
    <p>
        Suspendisse rutrum vestibulum odio, sed venenatis purus condimentu
    </p>

    <h6>Six</h6>
    <p>
        Sed quis malesuada mi. Nam id purus quis augue sagittis pharetra.
    </p>

</body>

</html>

```

Notice that `<h1>`, `<h2>`, and `<h3>` denote different levels of headings.

- We can also create lists within HTML:

```

<!DOCTYPE html>

<!-- Demonstrates (ordered) lists -->

<html lang="en">
  <head>
    <title>list</title>
  </head>
  <body>
    <ol>
      <li>foo</li>
      <li>bar</li>
      <li>baz</li>
    </ol>
  </body>
</html>

```

Notice that the `` tag creates an ordered list containing three items.

- We can also create a table in HTML:

```

<!DOCTYPE html>

<!-- Demonstrates table -->

<html lang="en">
  <head>
    <title>table</title>
  </head>
  <body>
    <table>
      <tr>
        <td>1</td>
        <td>2</td>
        <td>3</td>
      </tr>
    </table>
  </body>
</html>

```

```

        </tr>
        <tr>
            <td>4</td>
            <td>5</td>
            <td>6</td>
        </tr>
        <tr>
            <td>7</td>
            <td>8</td>
            <td>9</td>
        </tr>
        <tr>
            <td>*</td>
            <td>0</td>
            <td>#</td>
        </tr>
    </table>
</body>
</html>

```

Tables also have tags that open and close each element within.

- Images can also be utilized within HTML:

```

<!DOCTYPE html>

<!-- Demonstrates image -->

<html lang="en">
    <head>
        <title>image</title>
    </head>
    <body>
        <!-- https://www.harvard.edu/ -->
        
    </body>
</html>

```

Notice that `src="harvard.jpg"` indicates the path where the image file can be located.

- Videos can also be included in HTML:

```

<!DOCTYPE html>

<!-- Demonstrates video -->

<html lang="en">
    <head>
        <title>video</title>
    </head>
    <body>
        <!-- https://www.harvard.edu/ -->
        <video autoplay loop muted playsinline width="1280">
            <source src="halloween.mp4" type="video/mp4">
        </video>
    </body>
</html>

```

```

    </body>
</html>

```

Notice that the `width` attribute defines the width of the video.

- You can also link between various web pages:

```

<!DOCTYPE html>

<!-- Demonstrates link -->

<html lang="en">
  <head>
    <title>link</title>
  </head>
  <body>
    Visit <a href="image.html">Harvard</a>.
  </body>
</html>

```

Notice that the `<a>` or *anchor* tag is used to make `Harvard` a linkable text.

- Meta tags are used to hold information about the data within the HTML file. Consider the following:

```

<!DOCTYPE html>

<!-- Demonstrates responsive design -->

<html lang="en">
  <head>
    <meta name="viewport" content="initial-scale=1, width=device-width">
    <title>meta</title>
  </head>
  <body>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus conva
  </body>
</html>

```

Notice this set of `meta` attributes makes this page mobile-friendly.

- There are numerous `meta` key-value pairs that you can use:

```

<!DOCTYPE html>

<!-- Demonstrates Open Graph tags -->

<html lang="en">
  <head>
    <meta property="og:title" content="CS50">
    <meta property="og:description" content="Introduction to the intellect">
    <meta property="og:image" content="cat.jpg">
    <title>meta</title>
  </head>
  <body>

```



```
...
</body>
</html>
```

Notice that these key value pairs relate to the `title` and `description` of the web page.

- You can also create forms reminiscent of Google's search:

```
<!DOCTYPE html>

<!-- Demonstrates form -->

<html lang="en">
  <head>
    <title>search</title>
  </head>
  <body>
    <form action="https://www.google.com/search" method="get">
      <input name="q" type="search">
      <input type="submit" value="Google Search">
    </form>
  </body>
</html>
```

Notice that a `form` tag opens and provides the attribute of what `action` it will take. The `input` field is included, passing the name `q` and the type as `search`.

- We can make this search better as follows:

```
<!DOCTYPE html>

<!-- Demonstrates additional form attributes -->

<html lang="en">
  <head>
    <title>search</title>
  </head>
  <body>
    <form action="https://www.google.com/search" method="get">
      <input autocomplete="off" autofocus name="q" placeholder="Query" type="search">
      <button>Google Search</button>
    </form>
  </body>
</html>
```

Notice that `autocomplete` is turned `off`. `autofocus` is enabled.

- We've seen just a few of many HTML elements you can add to your site. If you have an idea for something to add to your site that we haven't seen yet (a button, an audio file, etc.) try Googling "X in HTML" to find the right syntax!

CSS

- `CSS`, or *cascading style sheet*, is a markup language that allows you to fine-tune the aesthetics of your HTML files.
- In your terminal, type `code home.html` and write code as follows:

```
<!DOCTYPE html>

<!-- Demonstrates inline CSS with P tags -->

<html lang="en">
  <head>
    <title>css</title>
  </head>
  <body>
    <p style="font-size: large; text-align: center;">
      John Harvard
    </p>
    <p style="font-size: medium; text-align: center;">
      Welcome to my home page!
    </p>
    <p style="font-size: small; text-align: center;">
      Copyright &#169; John Harvard
    </p>
  </body>
</html>
```

Notice that some `style` attributes are provided to the `<p>` tags. The `font-size` is set to `large`, `medium`, or `small`. Then `text-align` is set to `center`.

- While correct, the above is not well-designed. We can remove redundancy by modifying our code as follows:

```
<!DOCTYPE html>

<!-- Removes outer DIV -->

<html lang="en">
  <head>
    <title>css</title>
  </head>
  <body style="text-align: center">
    <div style="font-size: large">
      John Harvard
    </div>
    <div style="font-size: medium">
      Welcome to my home page!
    </div>
    <div style="font-size: small">
      Copyright &#169; John Harvard
    </div>
  </body>
</html>
```

Notice that `<div>` tags are used to divide up this HTML file into specific regions. `text-align: center` is invoked on the entire body of the HTML file.

- It turns out that there is newer semantic text that is included in HTML. We can modify our code as follows:

```
<!DOCTYPE html>

<!-- Uses semantic tags instead of DIVs -->

<html lang="en">
  <head>
    <title>css</title>
  </head>
  <body style="text-align: center">
    <header style="font-size: large">
      John Harvard
    </header>
    <main style="font-size: medium">
      Welcome to my home page!
    </main>
    <footer style="font-size: small">
      Copyright &#169; John Harvard
    </footer>
  </body>
</html>
```

Notice that the `header` and `footer` both have different styles assigned to them.

- This practice of placing the style and information all in the same location is not good practice. We could move the elements of style to the top of the file as follows:

```
<!-- Demonstrates class selectors -->

<html lang="en">
  <head>
    <style>

      .centered
      {
        text-align: center;
      }

      .large
      {
        font-size: large;
      }

      .medium
      {
        font-size: medium;
      }

      .small
      {
        font-size: small;
      }

    </style>
    <title>css</title>
  </head>
```

```

<body class="centered">
  <header class="large">
    John Harvard
  </header>
  <main class="medium">
    Welcome to my home page!
  </main>
  <footer class="small">
    Copyright &#169; John Harvard
  </footer>
</body>
</html>

```

Notice all the style tags are placed up in the `head` in the `style` tag wrapper. Also notice that we've assigned *classes*, called `centered`, `large`, `medium`, and `small` to our elements, and that we select those classes by placing a dot before the name, as in `.centered`

- It turns out that we can move all our style code into a special file called a *CSS* file. We can create a file called `style.css` and paste our classes there:

```

.centered
{
  text-align: center;
}

.large
{
  font-size: large;
}

.medium
{
  font-size: medium;
}

.small
{
  font-size: small;
}

```

Notice that this is verbatim what appeared in our HTML file.

- We then can tell the browser where to locate the CSS for this HTML file:

```

<!DOCTYPE html>

<!-- Demonstrates external stylesheets -->

<html lang="en">
  <head>
    <link href="style.css" rel="stylesheet">
    <title>css</title>
  </head>
  <body class="centered">
    <header class="large">
      John Harvard
    </header>

```

```
<main class="medium">
  Welcome to my home page!
</main>
<footer class="small">
  Copyright © John Harvard
</footer>
</body>
</html>
```

Notice that `style.css` is linked to this HTML file as a stylesheet, telling the browser where to locate the styles we created.

Frameworks

- Similar to third-party libraries we can leverage in Python, there are third-party libraries called *frameworks* that we can utilize with our HTML files.
- *Bootstrap* is one of these frameworks that we can use to beautify our HTML and easily perfect design elements such that our pages are more readable.
- Bootstrap can be utilized by adding the following `link` tag in the `head` of your html file:

```
<head>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap"
  <title>favorites</title>
</head>
```

- You can learn more about this in the [Bootstrap Documentation](https://getbootstrap.com/docs/4.1/getting-started/introduction/) (<https://getbootstrap.com/docs/4.1/getting-started/introduction/>).

JavaScript

- JavaScript is another programming language that allows for interactivity within web pages.
- JavaScript supports conditionals:

```
if (x < y)
{
}
else
{
}
```

- Variables are also supported:

```
let counter = 0;
```

- You can also increment:

```
counter++
```

- Loops are very similar to what you have seen before in C:

```
for (let i = 0; i < 3; i++)  
{  
  
}
```

- JavaScript allows you to dynamically read and modify the html document loaded into memory such that the user need not reload to see changes.
- Consider the following HTML:

```
<!DOCTYPE html>  
  
<!-- Demonstrates programmatic changes to style -->  
  
<html lang="en">  
  <head>  
    <title>background</title>  
  </head>  
  <body>  
    <button id="red">R</button>  
    <button id="green">G</button>  
    <button id="blue">B</button>  
    <script>  
  
      let body = document.querySelector('body');  
      document.querySelector('#red').addEventListener('click', function(  
        body.style.backgroundColor = 'red';  
      ));  
      document.querySelector('#green').addEventListener('click', function(  
        body.style.backgroundColor = 'green';  
      ));  
      document.querySelector('#blue').addEventListener('click', function(  
        body.style.backgroundColor = 'blue';  
      ));  
  
    </script>  
  </body>  
</html>
```

Notice that JavaScript listens for when a specific button is clicked. Upon such a click, certain style attributes on the page are changed. `body` is defined as the body of the page. Then, an event listener waits for the clicking of one of the buttons. Then, the `body.style.backgroundColor` is changed.

- Similarly, consider the following:

```
<!DOCTYPE html>  
  
<html lang="en">  
  <head>  
    <script>
```

```

// Toggles visibility of greeting
function blink()
{
    let body = document.querySelector('body');
    if (body.style.visibility == 'hidden')
    {
        body.style.visibility = 'visible';
    }
    else
    {
        body.style.visibility = 'hidden';
    }
}

// Blink every 500ms
window.setInterval(blink, 500);

</script>
<title>blink</title>
</head>
<body>
    hello, world
</body>
</html>

```

This example blinks a text at a set interval. Notice that `window.setInterval` takes in two arguments: 1) A function to be called and 2) a wait period (in milliseconds) between function calls.

- Consider the following:

```

<!DOCTYPE html>

<html lang="en">

    <head>
        <title>autocomplete</title>
    </head>

    <body>

        <input autocomplete="off" autofocus placeholder="Query" type="text">

        <ul></ul>

        <script src="large.js"></script>
        <script>

            let input = document.querySelector('input');
            input.addEventListener('keyup', function(event) {
                let html = '';
                if (input.value) {
                    for (word of WORDS) {
                        if (word.startsWith(input.value)) {
                            html += `<li>${word}</li>`;
                        }
                    }
                }
            });
        </script>
    </body>
</html>

```

```
        }  
        document.querySelector('ul').innerHTML = html;  
    });  
  
</script>  
  
</body>  
</html>
```

This is a JavaScript implementation of autocomplete.

- Interestingly, we can also geolocate using JavaScript:

```
<!DOCTYPE html>  
  
<html lang="en">  
  <head>  
    <title>geolocation</title>  
  </head>  
  <body>  
    <script>  
  
        navigator.geolocation.getCurrentPosition(function(position) {  
            document.write(position.coords.latitude + ", " + position.coor  
        });  
  
    </script>  
  </body>  
</html>
```

Notice that `navigator.geolocation` is used to `getCurrentPosition`. This will not work if your computer or browser does not allow for location tracking.

- The capabilities of JavaScript are many and can be found in the [JavaScript Documentation](https://developer.mozilla.org/en-US/docs/Web/JavaScript) (<https://developer.mozilla.org/en-US/docs/Web/JavaScript>).

Summing Up

In this lesson, you learned how to create your own HTML files, style them, leverage third-party frameworks, and utilize JavaScript. Specifically, we discussed...

- TCP/IP
- DNS
- HTML
- CSS
- Frameworks
- JavaScript

See you next time!

