This is CS50

CS50's Introduction to Computer Science

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Lecture 8

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

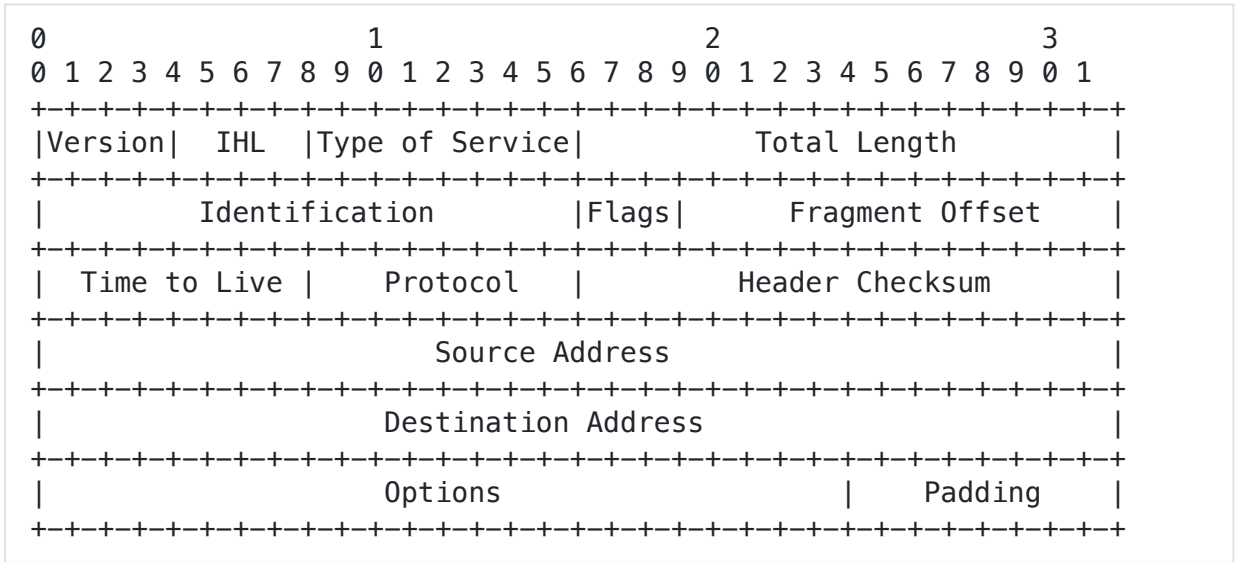
- The internet is a technology that we all use.
- Using our skills from previous weeks, we can build our own web pages and applications.
- The *ARPANET* connected the first points on the internet to one another.
- Dots between two points could be considered *routers*.

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. *Packets* of data are transferred from one router to another, from one computer to another.
- *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, implementing 128-bits, can accommodate far more computers!
- In the real world, servers do a lot of work for us.
- Packets are structured as follows:



- Packets are standardized. The source and destination are held within each packet.
- *TCP*, or transmission control protocol, helps keep track of the sequence of packets being sent.
- Further, TCP 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, a source IP address, a destination IP address, and a TCP port number are sent.
- These protocols are also used to fragment large files into multiple parts or 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 IP address 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 is simply a table or database that links specific, fully qualified domain names to specific IP addresses.

DHCP

- *DHCP* is a protocol that ascertains the IP address of your device.
- Further, this protocol defines the default gateway and nameservers your device uses.

HTTPS

- *HTTP* or *hypertext transfer protocol* is an application-level protocol that developers use to build powerful and useful things through the transfer of data from one place to another. *HTTPS* is a secure version of this protocol.
- 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` directory, and then visits the file named `file.html`.
- The `.com` is called a *top-level domain* that is used to denote the location or type of organization associated with this address.
- `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.

- For example, when issuing a GET request, your computer may send the following to a server:

  ```
  GET / HTTP/2
  Host: www.harvard.edu
  ```

  Notice that this requests via HTTP the content served on www.harvard.edu.

- Generally, after making a request to a server, you will receive the following in `Response Headers`:

  ```
  HTTP/2 200
  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.

- Via developer tools in your web browser, you can see all the HTTP requests when browsing to the above website.

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

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

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

- 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`, meaning that the website has permanently moved.

- 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 when they concern a product or application of your creation. This will be especially important for next week's problem set, and potentially for your final project!

HTML

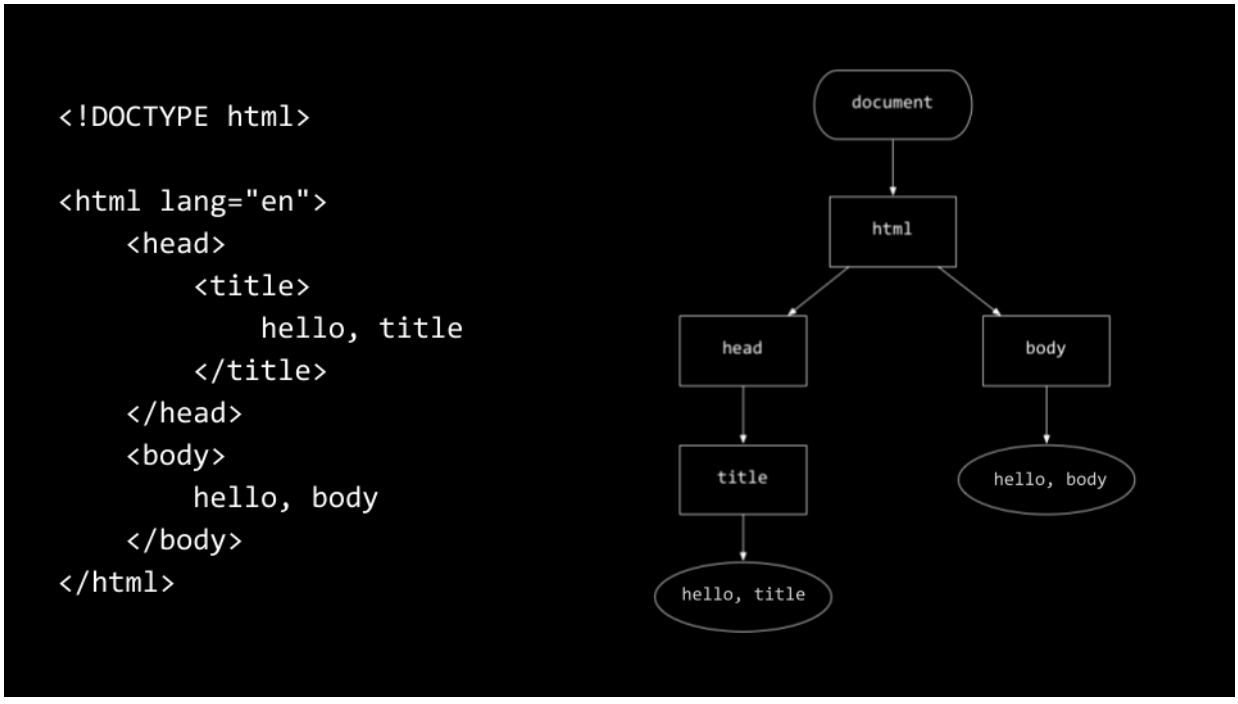
- *HTML* or *hypertext markup language* is made up of *tags*, each of which may have some *attributes* that describe it.
- In your terminal, type `code hello.html` and write code as follows:

```
<!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 served content is now available on a very long URL. If you click it, you can visit the website generated by your own code.
- When you visit this URL, notice that the file name `hello.html` appears at the end of this URL. Further, notice, based upon the URL, that the server is serving via port 8080.
- The hierarchy of tags can be represented as follows:



- Knowledge of this hierarchy will be useful later as we learn JavaScript.
- The browser will read your HTML file top to bottom and left to right.
- Because whitespace and indentation are 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. Viv
        </p>
        <p>
            Mauris ut dui in eros semper hendrerit. Morbi vel elit mi. S
        </p>
        <p>
            Aenean venenatis convallis ante a rhoncus. Nullam in metus v
        </p>
        <p>
            Integer at justo lacinia libero blandit aliquam ut ut dui. C
        </p>
        <p>
            Suspendisse rutrum vestibulum odio, sed venenatis purus conc
        </p>
        <p>
            Sed quis malesuada mi. Nam id purus quis augue sagittis phar
        </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. Viv
        </p>
```

```
        <h2>Two</h2>
        <p>
            Mauris ut dui in eros semper hendrerit. Morbi vel elit mi. S
        </p>

        <h3>Three</h3>
        <p>
            Aenean venenatis convallis ante a rhoncus. Nullam in metus v
        </p>

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

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

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

    </body>

</html>
```

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

- We can also create unordered lists within HTML:

```
<!DOCTYPE html>

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

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

Notice that the `<ul>` tag creates an unordered list containing three items.

- We can also create ordered 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 `<ol>` 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>
            <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. Also, notice the syntax for comments in HTML.

- Images can also be utilized within HTML:

```
<!DOCTYPE html>

<!-- Demonstrates image -->

<html lang="en">
    <head>
        <title>image</title>
    </head>
    <body>
        <img alt="photo of bridge" src="bridge.png">
    </body>
</html>
```

Notice that `src="bridge.png"` 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>
        <video controls muted>
            <source src="video.mp4" type="video/mp4">
        </video>
    </body>
</html>
```

Notice that the `type` attribute designates that this is a video of type `mp4`. Further, notice how `controls` and `muted` are passed to `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="https://www.harvard.edu">Harvard</a>.
    </body>
</html>
```

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

- 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="Q
            <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! Similarly, you can use cs50.ai (https://cs50.ai) to help you discover more HTML features!

Regular Expressions

- *Regular expressions* or *regexes* are a means by which to ensure that user-provided data fits a specific format.
- We can implement our own registration page that utilizes regexes as follows:

```
<!DOCTYPE html>

<!-- Demonstrates type="email" -->

<html lang="en">
    <head>
        <title>register</title>
```

```
    </head>
    <body>
        <form>
            <input autocomplete="off" autofocus name="email" placeholde
            <button>Register</button>
        </form>
    </body>
</html>
```

Notice that the `input` tag includes attributes that designate that this is of type `email`. The browser knows to double-check that the input is an email address.

- While the browser uses these built-in attributes to check for an email address, we can add a `pattern` attribute to ensure that only specific data ends up in the email address:

```
<!DOCTYPE html>

<!-- Demonstrates pattern attribute -->

<html lang="en">
    <head>
        <title>register</title>
    </head>
    <body>
        <form>
            <input autocomplete="off" autofocus name="email" pattern=".
            <button>Register</button>
        </form>
    </body>
</html>
```

Notice that the `pattern` attribute is handed a regular expression to denote that the email address must include an `@` symbol and a `.edu`.

- You can learn more about regular expressions from Mozilla's documentation (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions). Further, you can make inquiries to cs50.ai (https://cs50.ai) for hints.

CSS

- `CSS`, or *cascading style sheet*, is a markup language that allows you to fine-tune the aesthetics of your HTML files.
- CSS is filled with *properties*, which include key-value pairs.
- 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. Because everything inside `body` is a child of `body`, the `center` attribute cascades down to those children.

- It turns out that there are newer semantic tags 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 &#169; 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@5.3.3/dist/css/bo
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bo
    <title>bootstrap</title>
</head>
```

- Consider the following HTML:

```
<!DOCTYPE html>

<!-- Demonstrates table -->

<html lang="en">
    <head>
        <title>phonebook</title>
    </head>
    <body>
        <table>
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Number</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>Carter</td>
                    <td>+1-617-495-1000</td>
                </tr>
                <tr>
                    <td>David</td>
                    <td>+1-617-495-1000</td>
                </tr>
                <tr>
                    <td>John</td>
                    <td>+1-949-468-2750</td>
                </tr>
            </tbody>
        </table>
```

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

Notice how, when looking at a served version of this page, it's quite plain.

- Now consider the following HTML that implements the use of Bootstrap:

```
<!DOCTYPE html>

<!-- Demonstrates table with Bootstrap -->

<html lang="en">
    <head>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/cs
        <title>phonebook</title>
    </head>
    <body>
        <table class="table">
            <thead>
                <tr>
                    <th scope="col">Name</th>
                    <th scope="col">Number</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>Carter</td>
                    <td>+1-617-495-1000</td>
                </tr>
                <tr>
                    <td>David</td>
                    <td>+1-949-468-2750</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```

Notice how much prettier this website is now.

- Similarly, consider the following expansion of our search page created earlier:

```
<!DOCTYPE html>

<!-- Demonstrates layout with Bootstrap -->

<html lang="en">
    <head>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/cs
        <title>search</title>
    </head>
    <body>

        <div class="container-fluid">

            <ul class="m-3 nav">
                <li class="nav-item">
```

```
                    <a class="nav-link text-dark" href="https://about.g
                </li>
                <li class="nav-item">
                    <a class="nav-link text-dark" href="https://store.g
                </li>
                <li class="nav-item ms-auto">
                    <a class="nav-link text-dark" href="https://www.goo
                </li>
                <li class="nav-item">
                    <a class="nav-link text-dark" href="https://www.goo
                </li>
                <li class="nav-item">
                    <a class="nav-link text-dark" href="https://www.goo
                        <svg xmlns="http://www.w3.org/2000/svg" width="1
                            <path d="M1 2a1 1 0 0 1 1-1h2a1 1 0 0 1 1 1v
                        </svg>
                    </a>
                </li>
                <li class="nav-item">
                    <a class="btn btn-primary" href="https://accounts.g
                </li>
            </ul>

            <div class="text-center">

                <!-- https://knowyourmeme.com/memes/happy-cat -->
                <img alt="Happy Cat" class="img-fluid w-25" src="cat.gi

                <form action="https://www.google.com/search" class="mt-4
                    <input autocomplete="off" autofocus class="form-con
                    <button class="btn btn-light">Google Search</button>
                    <button class="btn btn-light" name="btnI">I'm Feeli
                </form>

            </div>

        </div>

    </body>
</html>
```

This version of the page is exceedingly stylized, thanks to Bootstrap.

- You can learn more about this in the Bootstrap Documentation (https://getbootstrap.com/docs/).

JavaScript

- JavaScript is another programming language that allows for interactivity within web pages.
- Consider the following implementation of `hello.html` that includes both JavaScript and HTML:

```
<!DOCTYPE html>

<!-- Demonstrates onsubmit -->

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

            function greet()
            {
                alert('hello, ' + document.querySelector('#name').value
            }

        </script>
        <title>hello</title>
    </head>
    <body>
        <form onsubmit="greet(); return false;">
            <input autocomplete="off" autofocus id="name" placeholder="N
            <input type="submit">
        </form>
    </body>
</html>
```

Notice how this form uses an `onsubmit` property to trigger a `script` found at the top of the file. The script uses `alert` to create an alert pop-up. `#name.value` goes to the textbox on the page and obtains the value typed by the user.

- Generally, it's considered bad design to mix onsubmit and JavaScript. We can advance our code as follows:

```
<!DOCTYPE html>

<!-- Demonstrates DOMContentLoaded -->

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

            document.addEventListener('DOMContentLoaded', function() {
                document.querySelector('form').addEventListener('submit
                    alert('hello, ' + document.querySelector('#name').va
                    e.preventDefault();
                });
            });

        </script>
        <title>hello</title>
    </head>
    <body>
        <form>
            <input autocomplete="off" autofocus id="name" placeholder="N
            <input type="submit">
        </form>
```

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

Notice that this version of the code creates an `addEventListener` to listen to the form `submit` being triggered. Notice how `DOMContentLoaded` ensures that the whole page is loaded before executing the JavaScript.

- We can advance this code as follows:

```
<!DOCTYPE html>

<!-- Demonstrates keyup and template literals -->

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

            document.addEventListener('DOMContentLoaded', function() {
                let input = document.querySelector('input');
                input.addEventListener('keyup', function(event) {
                    let name = document.querySelector('p');
                    if (input.value) {
                        name.innerHTML = `hello, ${input.value}`;
                    }
                    else {
                        name.innerHTML = 'hello, whoever you are';
                    }
                });
            });

        </script>
        <title>hello</title>
    </head>
    <body>
        <form>
            <input autocomplete="off" autofocus placeholder="Name" type=
        </form>
        <p></p>
    </body>
</html>
```

Notice that the DOM is dynamically updated in memory as the user types out a name. If there is a value inside `input`, upon the `keyup` on the keyboard, the DOM is updated. Otherwise, default text is presented.

- 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', fu
                body.style.backgroundColor = 'red';
            });
            document.querySelector('#green').addEventListener('click',
                body.style.backgroundColor = 'green';
            });
            document.querySelector('#blue').addEventListener('click', fu
                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: A function to be called and a waiting period (in milliseconds) between function calls.

- Consider the following implementation of JavaScript that autocompletes text:

```
<!DOCTYPE html>

<html lang="en">

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

    <body>

        <input autocomplete="off" autofocus placeholder="Query" type="te

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

        </script>

    </body>
</html>
```

This is a JavaScript implementation of autocomplete. This pulls from a file (not pictured here) called `large.js` that is a list of words.

- The capabilities of JavaScript are many and can be found in the JavaScript Documentation (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
- Regular expressions.
- CSS
- Frameworks
- JavaScript

See you next time!