Intro to Web Development

compileHer x Uncommon Hacks

HTML5

Good resource for a lot of elements: [W3Schools.com – HTML Basic Examples]

General Info on Writing HTML Code

- You always need to start a document with
  - `<!DOCTYPE HTML>`
- Then, put all your html inside two tags like this:

```
<html>
<body>
**your content**
</body>
</html>
```

- This gives you an idea of how HTML pages are structured in general:
  - elements are identified with <> brackets. The first time it's just the element name, the second time there's a / before the element name.
  - You want to nest HTML element
- Attributes - [Common HTML Attributes]
  - Images: `src`, `width`, `height`, `alt`
    - `<img src="myimg.png" width="200px" height="200px"/>`
  - Link: `href`
  - `style`, `title`

Useful tags to know:

- **p (paragraph)**:
  - `<p> YOUR CONTENT </p>`
- **h1-h6 (headings)**
  - `<h1> YOUR CONTENT </h1>`
  - etc.
  - h1 is the largest, and they decrease in size from there
- **a (link)**
  - `<a href="https://www.yourlink.com">Hyperlink Text</a>`
- **img**

- More info: HTML Images link

```
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

- Note that images are encapsulates all in one tag (<img>)
- Alt text: alternative description: helpful for accessibility and for descriptions in the case that an image doesn't properly load
- Where the image is located:
  - Can be in the same directory your code exists in, which is common. Then, just specify the path relative to the file your code is in.
  - You can use the url of an image, like so (be careful of copyrights depending on the project!):

```
<img src="https://www.w3schools.com/images/w3schools_green.jpg" alt="W3Schools.com">
```

- **hr (horizontal line)**
  - Full Reference HTML hr tag
  - `<hr>` on its own gives a horizontal line between content blocks
- **ul, ol (unordered + ordered list), li (list element)**
  - Full Reference: w3schools HTML Lists Tutorial
  - **Unordered List:**
    - inside brackets, with each element inside brackets
    - Example Code:

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

    - Output of Above Code:
      - Coffee
      - Tea
      - Milk
  - **Ordered List:**
    - Example Code

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

        - Output of Above Code

1. Coffee
2. Tea
3. Milk

- **Tables**: table, tr (row), th (header), td (cell)
    - Full Reference: W3Schools HTML Tables
    - Example Code:

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

    - Output of Above Code:

Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94
John	Doe	80

        -
    - Each row is defined within a <tr></tr> set of tags, each header within a <th></th> set of tags, and each element within a <td></td> set of tags
    - The whole thing is enclosed within a <table> element, and the width property refers to the percent of the width of the enclosing container the table takes up. To understand this more, look at the layout info below (because this is technically CSS, they're just doing it inline with the HTML via the "style" property).
    - See above link for more advanced formatting ideas
- **Div**
    - A set of tags define a section of a document in CSS. This doesn't change the way your HTML is shown on the page, but they can be helpful for logically breaking up code and for applying certain CSS styling rules only to a specific section of a page.

Tools

- View Source
- Inspector / DevTools - Javascript console, styling, HTML tags
    - Can select an object and see/edit its properties

CSS3

Some Good Examples:

[The Best CSS Examples and CSS3 Examples]

How to Connect CSS to HTML & Intro to CSS

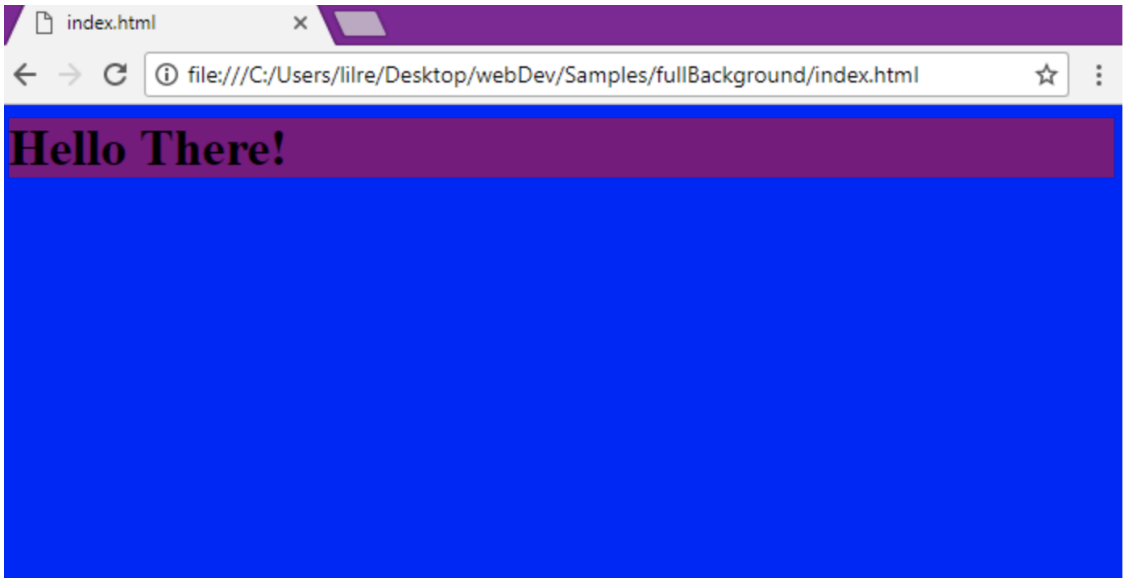
- [HTML Styles CSS]

Elements and Properties:

- Things like body, headers, and paragraphs are HTML elements. Things like “background-color” or “width” are properties that can be specified with CSS to determine how these elements will be styled in the final page. At the most basic level, you can specify properties for all elements of a certain type.

```
body {
    background-color: blue;
}
h1 {
    background-color: purple;
}
```

Output:



Note here that you can open an HTML file (which is linked with a CSS file as we described above) in a web environment simply by saving it as .html and opening it in a browser (typically the default option).

Some Good Basic Properties to Know

- **Colors**
  - What you can set:
    - Depending on the element, you can set background-color, border, color, etc.
  - How to specify color: CSS Colors
    - You can use RGB, Hex, HSL, RGBA, HSLA, and color key words. See the above link for more info!
    - Example: `background-color:rgba(255, 99, 71, 0.5);`
    - The “a” corresponds to opacity.
  - Good resources of named colors: Color Names — HTML Color Codes and Web colors
- **Borders**
  - Full Reference: W3S – CSS Borders
  - For a given element, e.g., a paragraph, you can specify:
    - border-width
      - typically in px, that is, pixels
      - One value makes all sides that width
      - Two values: first corresponds to top and bottom, second to left and right
      - Four values: correspond to top, right, bottom, left respectively
    - border-color
    - border-style
      - dotted, solid, dashed, etc.
    - border-radius: an optional value that specifies, in pixels, the radius of the border corners. This can create rounded borders.
- **Fonts**
  - Specified via the font-family property
  - Typically, you want to specify increasingly generic fonts as a sort of fallback: if your first preference isn’t supported by whatever platform someone is accessing your page on, you want a backup to revert to.
  - E.g.:

```
.p1 {
  font-family: "Times New Roman", Times, serif;
}
```

  - Make sure to put multi-word font names in quotations

    - A good resource of common fonts: Fonts
- **Styling text:** CSS Text
- **Alignment**
    - Full Reference: CSS Layout - Horizontal & Vertical Align
    - Horizontally centering: you can use
        - `margin: auto`
        - as discussed above
        - If you don't set the width of an element, it will typically stretch to the margins of its container, and thus the centering won't matter
    - As seen in the Text link above, you can center text in an element with:
        - `text-align: center`

Selectors

- A good reference: CSS Selectors Reference
- The above code was an example of a "grouping" selector: all of a certain type of element are grouped together and given a set of style rules (e.g., p, or h1). This is also called selecting by tag name.
- What if you wanted just one paragraph to be a certain color, not all of them?
    - You can give that paragraph a unique ID in your HTML code, and then refer to that ID in the CSS, like so:
    - In HTML:

```
<p id="para1">Hello World!</p>
```

    - In CSS:

```
#para1 {
  text-align: center;
  color: red;
}
```

- What if you wanted a certain collection of paragraphs to have one color, but not all of them?
    - You can use classes!
    - You can specify the class of an HTML element like so:
        - `<h1 class="center">Red and center-aligned heading</h1>`
    - When you want to style for a class, you can apply a style to all elements of a class, like so:

```
.center {
  text-align: center;
  color: red;
}
```

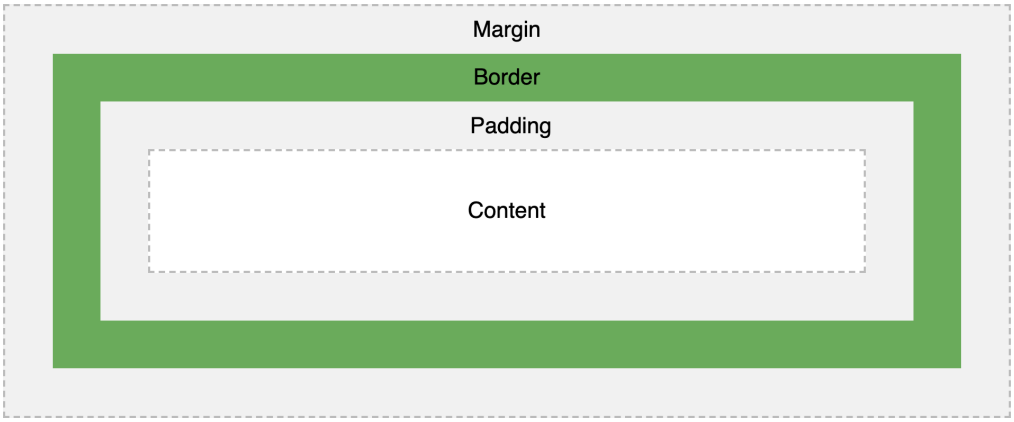
    - Or to just a specific element within a class, like so:

```
h1.center {
  text-align: center;
  color: red;
}
```

- Note that if you have multiple rules that apply to the same element, the most specific rule applicable to any one element will win out (e.g., if you have the background color set as blue for all paragraph elements, but have one paragraph element with ID p1 and you set that as purple, it'll be purple even while all the others are blue).
- Note that you can also put one HTML element in multiple classes, and use multiple class selectors
- Universal Selector: the * selector in CSS will let you apply a rule to an entire page
- Game for learning selectors: [CSS Diner - Where we feast on CSS Selectors!]

Box Model:

- A model of how to organize elements on the page relative to each other
- Full Reference: https://www.w3schools.com/css/css_boxmodel.asp
- In the box model, each HTML element, like a paragraph, is surrounded by a "box". This box is composed of nesting layers like so:



- Example: a element could be styled like so (which is a way you can divide up sections of content in code organization and allow it to group your stylings accordingly)

```
div {
  width: 320px;
  padding: 10px;
  border: 5px solid gray;
  margin: 0;
}
```

- Note that the above example uses shorthand to specify multiple properties of the border in one line
- Similarly to border-width, you can specify one, two, or four values for width, padding, border, and margine
- Auto: this property can be helpful for centering an element in a way that stays consistent even as the amount of content in it may change. A good explanation: https://ishadeed.com/article/auto-css/
    - Note that a parent element is an element containing another element, e.g., a paragraph with a certain div means the div is the parent.

Visibility/Display

- CSS Layout - inline-block
- `display: none` turns an element invisible
- `display: block` is default
- `display: inline-block` does not add a line break (`block` does)

Page Layouts

- CSS can also be used to make more complex and responsive layouts that are useful for positioning elements on the page
    - This is preferred to using, for instance, HTML tables for positioning - HTML should be the content, CSS how it shows up

CSS-column

- most basic, best for fixed number of columns (though can vary based on device dimensions)

```
.container {
  column-count: 3;
}
```

- This `class="container"` would be applied to a parent element, and as children the html elements (<p>, <img>, etc) that you want in the 3 columns
- You can also set column-gap, width, etc: w3schools CSS Multiple Columns
- Mozilla docs: CSS Multi-column Layout - CSS: Cascading Style Sheets | MDN

Flexbox

- Good for more flexible layouts, particularly adjustments to content, device size, etc
- In the `.container` style, set `display: flex`; `flex-direction` (row, row-reverse, column, column-reverse) for the main direction the items will be laid out in

```
.container {
    display: flex;
    flex-direction: column;
}
```

    - Set `flex-wrap: wrap` if you want things to wrap if they overflow; or `nowrap` if you want them to be cut off / potentially scrollable
    - `align-items` - how fit on the cross axis (ie center; default `stretch` (to fill))
    - `justify-content` - how fit on the main axis (default `flex-start`)
    - On the flex items, you can set proportional sizes - ie `flex: 1` for one element and `flex: 2` for a second (of 2 total) will make the latter take ⅔ of the space
    - Can override the alignment (`align-items` value) of an item with `align-self`
- Mozilla docs: Flexbox - Learn web development | MDN
- Game: Flexbox Froggy - A game for learning CSS flexbox

Grid

- Good for more advanced layouts where you want to be very precise about where each element is
- Set `display: grid` in the container style
- `grid-template-rows` and `grid-template-columns` to lay out what rows/columns
    - `1fr 1fr 1fr` would set 3 columns/rows of equal size (will grow/shrink with the page size), can also do specific hard-coded width/height values
- On individual items - need to set start and end row and column for each
    - Can set each explicitly (`grid-row-start`, etc), or shorthand:
    - `grid-row: 1 / span 2` for an element starting in row 1 and spanning 2 rows
    - These are numbered starting at 1
- Mozilla docs: Basic Concepts of grid layout - CSS: Cascading Style Sheets | MDN
    - CSS Grid Layout - CSS: Cascading Style Sheets | MDN
- Game: Grid Garden - A game for learning CSS grid

Flexbox or grid?

- Do I only need to control the layout by row or column? – use a flexbox
- Do I need to control the layout by row *and* column? – use a grid
- From content out — flexbox
    - An ideal use case for flexbox is when you have a set of items and want to space them out evenly in a container
    - Lets the size of the content decide how much individual space each item takes
    - If the items wrap onto a new line, they will work out their spacing based on their size and the available space on that line
- From layout in — grid
    - You create a layout and then you place items into it, or you allow the auto-placement rules to place the items into the grid cells according to that strict grid
    - Not based on content size

Responsive Design

- media queries

```
@media screen and (max-width: 800px) {
  /* special styling for narrow screen*/
}
```

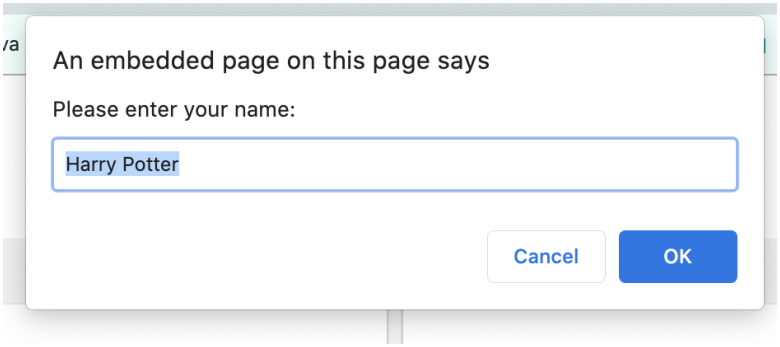
- Using media queries - CSS: Cascading Style Sheets | MDN
- Responsive design - Learn web development | MDN

Accessibility

- screen readers, visual alerting, and captioning
- Handling common accessibility problems - Learn web development | MDN
- Html for the elements/content, css for the styling!
- Use the built in tags from HTML5 in the ways they're meant to be used
- `alt` tags for images

Javascript

- **How To Connect Your Javascript**
  - Similarly to connecting CSS options you can:
    - put in <script> tags (JavaScript Where To) OR
    - separate script file which is included in script tag (HTML script src Attribute)
  - **Displaying Info**
    - console.log: logs to the web console console.log() - Web APIs | MDN
      - What's the web console? On Chrome: through DevTools: Console overview
      - Other browsers: check out the second post in this thread: javascript - What is console.log?
    - alert: JavaScript Popup Boxes: Alert Box
    - Alerts look like this (example from above reference--they don't have to allow someone to enter text though):



- **Event Handlers**
  - E.g., what do you want to do when a user presses a button? Event handlers allow you to attach javascript code, the "what happens" to HTML elements--e.g., the button.
  - A Good Event Handler Reference: Event Handlers (onclick, onkeypress, onfocus)
- **ES6** (useful new features / things to know)
  - let & const - ES6 In Depth: let and const
  - arrow functions - ES6 In Depth: Arrow functions
  - New methods on Arrays, maps & sets
    - Array - JavaScript | MDN
    - Map - JavaScript | MDN
    - Map - JavaScript | MDN
- **DOM manipulation**

- ie using selectors to get elements, modify attributes, etc.
  - Methods for Accessing Elements in the DOM File with JavaScript
- `getElementById` (select by id), `querySelector` (using CSS style selectors), `setAttribute`, `addEventListener` (to listen for click or similar)
- Ex:

```
let button = document.querySelector("#button")
button.setAttribute("disabled", "") // set disabled to
true
button.addEventListener("click", function () {
     // onclick action here
})
```

- Can also add or remove nodes
  - Create, Insert, Replace, and Delete DOM Nodes with JavaScript

```
const firstDiv = document.createElement("div")
const text = document.createTextNode("hello i'm a div")
firstDiv.appendChild(text)
document.getElementById("root").prepend(firstDiv)
// first child of root is firstDiv
```

Requests and Asynchronous Javascript

- **XML HTTP request** = one way of fetching data - Using XMLHttpRequest - Web APIs
  - be careful of cross origin requests (to a different domain) - CORS - Cross-Origin Resource Sharing (CORS) - HTTP

```
var request = new XMLHttpRequest();
// open the request
try {
  request.open("GET", url);
} catch(e) {
  console.log("Bad request (invalid URL)\n" + url);
}

// event handler for successful response
request.addEventListener("load", handleSuccess);
// can add other event handlers (ie errors)

// send the request
request.send();
```

- **Basic asynchronous Javascript** - timers or intervals
  - `setTimeout` & `setInterval` - some code is executed after certain amount of time has passed, or every x amount of time
    - Will enter separate queue (other code may be executed in the meantime)

    - WindowOrWorkerGlobalScope.setTimeout() - Web APIs | MDN
    - Window setInterval() Method

- **Promises** - for handling async dependencies better
    - Allows segment of code to wait for asynchronous code to finish, then executes (allows data dependencies without errors)
    - Promise - JavaScript | MDN
    - Using Promises - JavaScript | MDN
    - Ex:

```
new Promise((resolve, reject) => {
    console.log('one thing')
    resolve()
}).then(() => {
    console.log('another thing')
}).then(() => {
    console.log('a third thing')
})
```

    - Fetch - a better (promise-based) way to get data
        - Using Fetch - Web APIs | MDN

```
fetch(url, {
    method: 'GET',
    headers: {
        'Content-Type': 'application/json'
    },
})
.then(response => response.json())
.then(data => console.log(data))
```

Databases (MySQL)

Basic Commands:

- Reference for basic commands: MySQL Commands
- Note that SQL commands have a semicolon after them!
- Create a database:
    - `create database [database name];`
- See all created databases
    - `show databases;`
- Start using a particular database (must have already created it)
    - `use [database name];`
- See all the tables in a database you're currently using
    - `show tables;`

- See the schema of a table (i.e., name of attributes and what type each is)
  - `describe [table name];`
- See all the entries in a table:
  - `select * from [table name];`
- Create table
  - The first attribute is typically a primary key, which you specific with the key phrase "`primary key`" as a form of a record (the rows of the table) ID, as seen in the example. If you want SQL to autogenerate them for you, such that you don't have to specify the id every time you insert a record, you can also include the key word "`auto_increment`," also seen in the first example.
  - `not null` means that a value has to be included when you insert a record
  - `default [value]` means that if a value isn't specified when something is inserted, it gets the default value
  - General structure of create table statement:

```
create table [tablename] (
      attribute1 type1,
      …
      attributeN typeN);
```

  - Example:

```
      create table [table name] (
            personid int(50) not null auto_increment
            primary key,
            firstname varchar(35),
            middlename varchar(50),
            lastname varchar(50) default 'doe');
```

- SQL data types: MySQL :: MySQL 8.0 Reference Manual :: 11 Data Types
- More on creating tables (includes info on foreign keys): MySQL CREATE TABLE Statement By Examples
- Insert statement general structure:
  - `insert into [tablename] (Attribute1, …, AttributeN) values (Value1, …. ValueN);`
  - Example:
    - `` `insert into books (Title, SeriesID, AuthorID) VALUES ("Lair of Bones", 2, 2);` ``
- References for different kinds of select statements, group by / aggregations, and joins: W3Schools SQL Tutorial
- PHP Prepared Statements: Prepared statements and stored procedures - Manual

---

Other Resources

- Python + flask for backend: Flask
- Encryption: use bcrypt - https://github.com/pyca/bcrypt/

- Resources on single page apps
    - This means serving just a single page, based on URL parameters etc, different content shows up; generally faster than needing to load a new page each time something happens
    - Single-page application
    - Static Files — Flask Documentation (1.1.x)
    - Web Storage API - Web APIs
    - Cookie - MDN Web Docs Glossary: Definitions of Web-related terms | MDN
    - History API - Web APIs | MDN

Frontend Frameworks

- Instead of adding/modifying elements in Javascript using the DOM manipulation methods, can tie HTML elements rendered to data manipulation + updates
    - Does your web app need a front-end framework?
- React - https://reactjs.org/
    - https://reactjs.org/tutorial/tutorial.html
    - https://reactjs.org/docs/thinking-in-react.html
    - https://reactjs.org/docs/lifting-state-up.html
    - https://reactjs.org/docs/conditional-rendering.html
- Angular - https://angular.io/
- Vue - https://vuejs.org/