## HTML Basics

- Start with <!DOCTYPE html>
- The whole page must be included in <html> </html>
- <head> </head>
    - Should include <title> </title>
    - May include <meta charset=”utf-8”>
- <body> </body>
    - <h1>, <h2>, etc are headers (goes up to 6).
    - <p> </p> are is normal text (p for paragraph)
        - <br> is used to denote a line break (exactly like /n)
- Whitespace in HTML is ignored, line breaks must be specifically included

## HTML Text Emphasis

- <em> </em> is used to denote emphasis in text (default: italics)
- <strong> </strong> is used to denote importance in text (default: bold)

## HTML Lists

- <ul> </ul> denotes an unordered list (default: bullet)
- <ol> </ol> denotes an ordered list (default: arabic numerals)
- Within a set of list tags, each instance of <li> </li> is a new list item
- A list can be denoted within list item tags to create a nested list

## HTML Images

- <img src="url here" alt="alt text here" width="200" height=”200”>

## CSS Basics

- CSS: Cascading Style Sheets
- CSS is not HTML, but is embedded in HTML
- <style> </style> tells the browser that the inside code is CSS, not html
    - These style tags must be within <head> </head>
- Style structure: identifier {attribute:value;}
    - Ex. h2 {color: #F4F1DE;} makes all of the h2 elements that color
    - The body identifier applies to the whole page (so the background color of the whole page can be changed using this identifier (and background-color)

## CSS Selecting by ID

- In order to style a specific HTML element (instead of all elements of the same type), an ID tag can be declared in the HTML tag of the element <type id=”id-name”> </type>
    - Ex. <p id=”piano-types”> </p>
- Then, CSS can be applied to the tag, so it styles all elements with that tag #tag {attribute:value}
    - Ex. #piano-types {background-color: brown;}
- The same ID cannot be used on multiple tags
- Spaces cannot be used in ID names

## CSS Selecting by class

- Classes assign a given HTML element to a specific “group” that may include other elements on the page that are similar, so that they may be styled together
- A class tag must also be declared in the HTML tag
    - Ex. <p class:”instructions”> </p>
- Then it can be styled with CSS: .class-name {attribute:value;}
- Spaces cannot be used in class names

## HTML Links

- <a> </a> denotes a hyperlink. Anything within the tags, including images, is what is displayed on the page
- In order to specify the url, use the href attribute: <a href=”url here”> </a>
    - Both absolute and relative urls can be used, but it’s best practice to use absolute ones
- Adding the attribute target=”_blank” in the <a> tag makes the link open in a new window

## HTML Internal Links

- Internal links can create a hyperlink from one element on the same page to another
- The target element most contain an id in its tag, and the href in the <a> tag must be in the form “#id”
    - Ex. <h2 id=”web-history”> </h2> → <a href=”#web-history”> </a>

## HTML Tables

- Tables are denoted by <table> </table>
- Every table has a header tag <thead> </thead> that contains header rows
- Every table has a body tag <tbody> </tbody>
- Within one of these tags is <tr> </tr>, which means table row
- <th> </th> refer to the header cells, and are found within <thead> </thead>. They contain labels
- <td> </td> refer to body cells, and are found within <tbody> </tbody>. They contain data

## HTML Comments

- Comments are denoted by <!-- comment here -->

## CSS font-family property

- font-family: sans-serif;    //or serif, cursive, fantasy, monoscape,
- Uses the default serif, sans serif font for the computer
- font-family: “helvetica”;   //use for any specific font
    - To specify a “backup font” for the computer if the specific one is not installed, type font-family: “helvetica”, sans-serif;
-

## CSS font-size property

- font-size: 23...;
    - Units: px (denoted 23px),
- Measured in ems by default, which is a relative unit (if the body text is 10px and the titles are 2em, the titles will be 20px)
    - Body text is 1em by default

## CSS font styles and shorthand

- font-weight: bold;
- font-style: italic;
- text-decoration: underline;
- Shorthand: font: italic 13px fantasy;

Complete list of CSS Properties

## CSS Inheritance

- Many text styles are inherited
    - Ex. an property that is applied to <body> </body> will also be applied to <h1> </h1> inside of the body tags
- A more specific rule will overwrite an inherited property

## Web Development Tools

- Inspect element can be used to view the HTML code of a page and locate specific elements within that code
- There is also a window that allows you to view the CSS properties of the element
- Online editors: JSBin, Repl.it, Glitch, CodePen
- Desktop editors: Visual Studio Code, Atom, Sublime text, Notepad++
- Command line editors: Emacs, Vim
- You can host a website on a server so that it can be viewed anywhere and buy a domain using a domain registrar
- Github pages can be used to host websites for free (with a url ending in .github.io)

## CSS Grouping elements

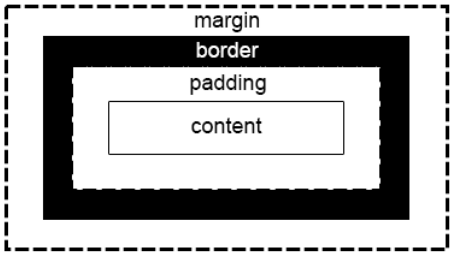
- <span> </span> is used to group selections of text (subsections within an element)
    - Inline element (does not create new lines)
- <div> </div> is used to group multiple elements together
    - Block element (creates new lines)
- Both of these can have ids and classes

## CSS Width, height, and overflow

- By default, <div> </div> will take up the entire page width
- Can be changed with the width attribute
    - width: 300px;
    - width: 70%;      //of available space
- The height of a <div> </div> can also be changed
    - height: 300px;
    - height: 70%;
- If the height of a div is too small for the content, it will overflow
    - overflow: visible;     //text spills past div (default)
    - overflow: hidden;      //text gets cut off at div border
    - overflow: auto;        //cuts off text, creates scroll bars
    - overflow-y and overflow-x can be set independently
- This can be applied to all elements, not just <div> </div>

## CSS Box model

- Every element on a webpage is considered a box by the browser



  - margin: 15px;     //all sides
  - margin: 15px 0px 10px 6px;  //trbl
  - margin-right, margin-bottom, etc
  - margin: auto;     //centered on page
  - border: 1px solid rgb(0, 0, 0);
    - //thickness type color
  - Border sides can be specific as well
  - padding: 15px;

## CSS Position

- Static positioning: normal positioning of elements (inline l → r)
- position: relative;     //normal position with some offset
  - top: , bottom: , left: , right:     //value of offset on each side
- position: absolute;     //absolute position on the page in px
  - top: , left:      //usually used to define positions
  - Relative to the top of the page (will move offscreen)
- position: fixed;        //defined in terms of the window, not the page
  - Won’t move when the page scrolls
- z-index: 1;     //a higher z-index makes the element appear on top

## CSS Floating elements

- Sometimes elements need to “wrap around” other elements (often text and images)
- float: left;     //is left of the other elements on the page
- When a <div> </div> is floating, a width should be specified because the element will otherwise try to take up the whole page
- clear: left;     //left, right, or both
  - Prevents it from wrapping around other elements

## Using multiple CSS classes

- Multiple classes can be added to a single tag
- They can be added with a space in the class string
  - Ex. <p class = “fact warning”>

## Combining CSS class and element selectors

- Element selectors and class selectors can be combined to further specify the elements to select
  - Ex. h1.warning {} selects all paragraph elements with the class warning

- Syntax: tagname.classname

## CSS descendant selectors

- Tags, and classes can be selected by their parent tag (descendant selector)
- Syntax: parentelement .class {}     //space is very important
    - Ex. p .explanation {}     //assume the tag with the explanation class is inside a <p> </p> tag
- Styles can also be applied to specific types of nested tags
    - Ex. li strong {} selects <strong> </strong> elements inside <li> </li> elements
    - More than two elements can be added (ex. ul li em)

## Grouping CSS selectors

- CSS selectors can be grouped to the same rules can be applied to multiple groups of elements
- Syntax: element1, element2 … {}
    - Ex. h1, h2 {}

## CSS dynamic pseudo-classes

- Dynamic pseudo-classes are used to select elements by something external, like a visited vs unvisited link
- Syntax: element/class:activity {}
    - Ex. p:hover {}
- Pseudo-classes: link, visited, hover, active (if the user is interacting with the element), focus (often activated when tabbing around the page)
- Similar pseudo-classes should be grouped
    - ex. a:hover, a:focus, a:active {}

## CSS specificity

- If there is no conflict, the browser will apply every style that applies to a given element (from multiple selectors)
- However, if they conflict, the one that gets rendered depends on the specificity
- ID is more specific than a class, which is more specific than an element
- If two rules are equally specific, the later one wins

## Using inline CSS styles

- A CSS style can be applied directly in the HTML code (inline style)
- This is done using a style attribute inside the HTML tag
- Ex: <h1 style=”background: salmon; color: white;”> </h1>
- Often used for first prototyping a web page, or during more restricted environments like emails

## Using external stylesheets

- Sometimes, many pages on the same website have the same style. This can be done efficiently using an external style sheet
- The <link> </link> tag must be used within <head> </head>:
  <link rel="stylesheet" type="text/css" href="linktostylesheet.css">

## HTML Validation

- Browsers will try to display your page even if there are errors in your code, so you might not know about the,
- W3C Markup validation service can check if you html is valid
    - Validator.w3.org

## Putting JS in a Webpage

- <head> </head> is loaded first, so we should include the <style> </style> inside so the page always loads the style correctly
- To add javascript to a page, we use the tags <script> </script>
    - <script> </script> needs to be loaded last, so it should be at the bottom of <head> </head>
- JS console: console.log(“le string”);

## Document Object Model

- Browser converts all of the html and css into a DOM: a javascript object that contains the whole page
    - Stored in an object called document
    - Object can be accessed like any other using dot modifier
        - Ex. document.body.innerHTML
- This object can then be manipulated like any other JS object

## Debugging web pages with the browser console

- The console will show errors in JS code and print any console.log statements
- Ctrl + Shift + j opens the console on chrome

## Finding elements by ID

- Node = element = some piece of the webpage
- Elements can be accessed by HTML page tag
    - Ex. document.body
- Elements can be accessed by id
    - Ex. document.getElementById(“ID_HERE”);
- An element's inner HTML can be accessed and changed
    - Ex. document.getElementById(“dogs_title”).innerHTML = “cats”;
- The element needs to be stored in a variable (???)
    - Ex. var headingEl = document.getElementById(“heading”);

## Finding multiple DOM elements by tag or class name

- Elements can be accessed by tag
    - Ex. document.getElementsByTagName("span");
- Because there may be more than one element like this, it is stored in an HTMLCollection
- Work like arrays: access an individual element using its index
    - Ex. console.log(span_elements[0]);
    - For loop should be used to iterate through all the elements
- Elements can be accessed by class name
    - Ex. document.getElementByClassName(“CLASS_NAME”);

## Finding elements by CSS selector

- Elements can be accessed by CSS Selector: the CSS selector is passed as a parameter in the access function
    - Ex. document.querySelectorAll("p .animal");
- Returns a NodeList (not the same as array or HTMLCollection) that uses the same syntax as an array
    - Nodelist is static (doesn’t update if more elements are added)
- document.querySelector(“css_selector”); only returns one element

## Traversing the DOM

- DOM elements can also be accessed by traversing the DOM’s tree-like structure
- Possible accessors to use:
    - firstElementChild
    - lastElementChild
    - nextElementChild/nextElementSibling
    - previousElementChild/previousElementSibling
    - childNodes
        - Ex. document.getElementById(“dog_pic”).childNotes[i];
    - childElementCount
- These only work on element nodes, not text nodes

## Changing attributes

- Some tags cannot be changed with innerHTML because they are self-contained; their attributes need to be changed
- An attribute can be changed using the dot operator
    - Ex. myElement.scr = “replacement_link_here”;
    - Structure: elementName.attributeName = value;
- CSS attribute selector: fancy CSS selector
    - Ex. a[href*=”dog”]    //selects all links that contain “dog”

## Changing styles

- Styles can also be changed using a dot operator: every element has a style attribute, which in turn has every css attribute
    - Ex. myElement.style.color = “red”;    //two dot operators!
- A different convention is used for attribute names in JS and CSS
    - CSS: spaces are denoted by dashes     //background-color
    - JS: spaces are denoted by camelCase   //backgroundColor

## Changing CSS classes

- Instead of changing every CSS attribute, a class (with its own style rules) can be created in <style> </style> and then applied to an element using JS
    - Ex. myElement.className = “class_name_here”;    //styles applied
    - class is a keyword in JS, so it can’t be used like in HTML
- This replaces any other class the element may have. If this is to be avoided, a class can be added instead
    - Ex. myElement.className += “class_name_here”;
    - Ex. myElement.classList.add(“class_name_here”);    //new browsers

## Setting innerHTML and textContent

- innerHTML replaces the HTML inside of tags → HTML tags can be added to the string in order to add elements
    - Ex. myElement.innerHTML = “text <em>here</em>”;
- If only text is to be changed, textContent can be used
    - Ex. myElement.textContent = “text_here”;

## Creating elements from scratch

- New elements can be added to the page (instead of modifying existing ones)
    a. var catEl = document.createElement("img");    //creates new tag
    b. catEl.src = "link_here";              //sets src attribute value
    c. document.body.appendChild(catEl);        //adds element to body
- Can be appended as a child anywhere on the page
- Text nodes are the children of other nodes, but aren't themselves nodes
    a. var strongText = document.createTextNode("cat");   //new text node
    b. strongEl.appendChild(strongText);   //text node linked to strongEl
    c. myElement.appendChild(strongEl)       //adds strongEl to myElement

## Making webpages interactive with events

- Events and listeners are what allow webpages to be interactive
- Examples: buttons, text input, form validation, slideshows, galleries

## Adding an event listener

- <button> </button> is an HTML tag that creates a button (woah)
- An event listener can link an action with a function for interactivity
    a. create the element we want to make interactive
    b. find and store the element we want to listen to events on
    c. define the function that will respond to the event (called a callback function)
    d. add the event listener for the element and function
- In practice
    a. <button id="clicker">Boring button</button>
    b. var clickerButton = document.getElementById("clicker");
    c. var onButtonClick = function() {clickerButton.textContent = "Oh wow, you clicked me!";}
    d. clickerButton.addEventListener("click", onButtonClick);
- Event listener syntax
    a. myElement.addEventListener("eventName", functionToCall);
- An anonymous function is a function declared inline without a name
    a. clickerButton.addEventListener("click", <u>function() {clickerButton.textContent = "Oh wow, you clicked me!";}</u>);
- Event listeners can be removed by calling removeEventListener() on the same object with the same parameters

## DOM event types

- Full list of events that the browser can trigger
- **Mouse events** (MouseEvent): mousedown, mouseup, click, dblclick, mousemove, mouseover, mousewheel, mouseout, contextmenu
- **Touch events** (TouchEvent): touchstart, touchmove, touchend, touchcancel
- **Keyboard events** (KeyboardEvent): keydown, keypress, keyup
- **Form events**: focus, blur, change, submit
- **Window events**: scroll, resize, hashchange, load, unload

## Using the event properties

- The event information object is used by the browser to store information about the event that has just occurred
- The browser sends it as the first argument when it calls the function attached to the event
- By explicitly adding as a parameter, its attributes can be accessed inside of the function (in JS, adding extra parameters does nothing)
    - e or evt or event are often used to denote it
    - Ex. var myCallbackFunction(e) = {...}
- Some useful attributes
    - e.clientX and e.clientY record the x and y positions of the mouse

## Processing forms with events

- Forms are used to collect information and sent it to a server
- <input> is an HTML tag that specifies an input element
    - Ex. <input id=”name” type=”text”>
    - To get what was entered into the form, use a dot operator: inputElement.value    //element can be stored or accessed from DOM
- <select> </select> is an HTML tag that specifies a dropdown selector, <option> </option> is used to mark options
    - Ex. <select id="lang"> <option value="en">English</option> <option value="es">Spanish</option> <select>
    - To get what was entered into the form, use a dot operator: inputElement.value    //element can be stored or accessed from DOM
- <label> </label> defines a label for various input elements

## Preventing default behavior of events

- Sometimes, you want to prevent default behavior (for example, opening a link in a new tab or window if you want it to do something on a page)
- Calling preventDefault on the event object in the callback function prevents this behavior

- Ex. var onOhNoesClick = function(e) { e.preventDefault(); ... };

## The window object

- The window object is another object supplied by the web browser
- It contains a lot of information
    - window.location.href          //url information
    - window.navigator.userAgent    //what browser the user is using
    - window.outerWidth, window.outerHeight    //dimensions of page
- window is the default global variable, so all of these properties can be accessed without using window.
    - Ex. navigator.userAgent
- When a new variable is created, it gets stored inside of the window object (so variables shouldn't have any of the same names as the reserved window variables like location)

## Animating DOM with setInterval

- window.setInterval(callback, millis) calls a certain function at a certain interval, which can be used to incrementally animate an object
    - Ex. window.setInterval(countdown, 200)
- Presumably, the callback function would somehow modify the html or css of the page to create an animation
- window.setTimeout(callback, millis) only runs the callback once, but specifies how long to wait before it will be called
    - Useful for a disappearing element
- window.clearInterval(interval) stops an occurrence of setInterval
    - In order to specify which one to stop, store the output of the original setInterval function in a variable, and pass that variable as a parameter into clearInterval

## Animating styles with requestAnimationFrame

- Using setinterval can be inefficient because it always runs, even if the window is not being rendered by the browser
- Solution: window.requestAnimationFrame(function)
    - This will run the callback function every time the browser renders a frame of the page (more efficient)
    - Speed can be altered by changing the function itself, or by storing the current runtime of the animation and changing according to that
- Only available on newer browsers (IE 10+)

## Animating styles with CSS animations

- In new browsers, animations can be programmed in CSS without any JS
- Uses a keyframe-ish system of **from** and **to** states
- To declare: @keyframes aName { from {properties} to {properties} }
- To use: selector { animation-name: name; animation-duration: 10s; }
    - Animation and duration are CSS properties like any other
- When using a dynamic pseudo-class, the transition property can be used to interpolate between the base and active state
    - Syntax: selector { property:value; transition: property 10s interpolationMethod }
    - Interpolation method can be linear, etc

## Using JS Libraries

- Libraries can be hosted externally or locally (different URL type)
- A locally hosted library can be tweaked and adapted
- A library that handles all aspects of a website (UI, data, DOM manipulation, etc) is called a framework