**HTML and CSS Library**

**HTML LIBRARY**

**General Formatting**

* <!--text for reference goes between these symbols-->
* <!DOCTYPE html> as the first line defines the code as html5
* After the doctype, everything should be included between <html></html>
* The head section <head></head> will not display on the page
  + The page <title></title> should go in the head and will display at the top of the window/tab
  + The head should also include meta data <meta /> to help search engines and caching behavior
  + The <meta /> section combines name=”” with content=””
    - Ex. <meta name=”keywords” content=”scuba, flounder, reproduction” />
      * Each entry (name/content pair) goes in a <meta /> element
    - Meta names should include description, keywords, & robots
    - Robots tells whether search engines should add that page to results
      * Name=”robots” content=”noindex” will not add page to search results
      * Content=”nofollow” will add this page but not any pages it links to
    - The http-equiv=”” and content=”” attributes are also used in <meta /> elements
      * Use <meta http-equiv=”author” content=”page author name” />
      * Use <meta http-equiv=”pragma” content=”no-cache” /> to keep page from caching
      * Use <meta http-equiv=”expires” content=”Sun, 08 Jul 2018 23:59:59 GMT” /> for a cache to expire at a given time
* Pages then have a body <body></body> and things here will display on the page
* Use a <header></header> within the body to show a page’s header usually including the title
* Use <nav></nav> to set apart page navigation (helps screen readers)
* Use a <footer></footer> to show page links and copyright info at the bottom of the page
* Use the <main></main> tag to outline the page’s main content (only one main section, main section should be in the body, but otherwise not a child to anything)
* Group multiple elements using <div></div> for css styling and page organization within the body or header
  + Putting multiple block style elements within a div makes them easier to position around each other
* Use <article></article> to surround standalone content that doesn’t rely on other things on the page (like an article)
* Use <aside></aside> within an article to provide related info that is not essential to the article or use it on its own for info related to the page that is not essential to the page (like additional links, list of recent posts…)
* Use <section></section> to distinguish between different sections that are parts of a whole
* Use the <span></span> element to style/distinguish in-line text, usually with class or id attributes
* Give elements id's for css and linking <h1 id="top"></h1> or <div id="services"></div>
  + Id’s should always be unique
* Elements can also be assigned a class=”” to quickly format multiple elements using css
  + Multiple elements can be the same class
  + An element can have more than one class, just separate with a space within the “”
* Link to id's using hashtag <a href="#top">Top</a>
* <h1></h1> is the primary header, and they go to <h6></h6>
* Paragraphs are enclosed by <p></p>
* There is a <time></time> element that can be <time datetime=”YYYY-MM-DD”>display text</time>
  + This helps screen readers and accessibility
* <hgroup></hgroup> can be used to group multiple h elements such as a title and subtitle (not necessary, but you may see it)

**Link to CSS and JavaScript Files**

* See “Linking to CSS File” section below
* Add the code <script src=”director/filename.js”></script> where you want the script to run in your page (not necessarily in the head, could be in a <p> element)

**Text Formatting**

* Bold words <b></b> and italicized words <i></i>
* Strong <strong></strong> words will display in bold and be read with strong emphasis by text readers
* Emphasis <em></em> words will display in italics and be read with subtle emphasis by text readers
* Superscripts <sup></sup> and subscripts <sub></sub>
* Line breaks <br /> are self closing
* Horizontal rule (line) <hr /> are self closing and break displaying a line
* The <blockquote cite="webaddress"></blockquote> element is used to set apart a longer quote (indented)
* The <q></q> is supposed to add quotation marks, but older internet explorer doesn't
* Use the <abbr></abbr> element for abbreviations/acronyms, and a title element must be included <abbr title="Professor">Prof</abbr>
* Use the <cite></cite> element for citations. text is displayed italicized.
* Use the <dfn></dfn> element to show where you are providing a definition of a term
* Use the <address></address> element to specify the page author's contact info
* The <ins></ins> tag shows inserted text (underline) and <del></del> shows deleted text with a strikethrough
* Use the <s></s> strikethrough if you want the text to show without being deleted
* Escape characters are either special characters or characters used for html coding (html entities)
  + You must code these characters
  + < is less than or &lt; or &#60;
  + > is greater than or &gt; &#62;
  + & is &amp;
  + See <http://www.htmlandcssbook.com/extras/html-escape-codes/> for a list of escape codes

**Lists**

* numbered (ordered) lists <ol></ol> and bulleted lists <ul></ul> include list items <li></li>
* definition lists <dl></dl> contain definition terms <dt></dt> and definitions <dd></dd> and the definitions are indented to the terms
* nested lists are possible <li><ul><li><li/></ul></li>

**Links**

* Use the accesskey=”” attribute to specify one keyboard letter to quickly jump to an element
  + Use attribute with any element, but can be particularly useful with links
* Links take the format <a href="linkaddress">text to click</a> using an anchor <a></a> element with href attribute
* It is good practice to make the homepage name for a section "index.html"
* Link to a file in the same folder <a href="filename">link display text</a>
  + Make link display text descriptive for screen readers that only look for links
  + Avoid using “click here” or equivalent statements as the link display text
* Link to a child or grandchild folder, add the directory <a href="child/grandchild/filename.html">text to display</a>
* Make an image a link by nesting it within an <a></a> element
* Add "../" every time you need to go up a level (example going up two levels to homepage) <a href="../../index.html"Home Page"</a>
* Email links are setup like this <a href="mailto:name@domain.com">Email Address</a>
* Open links in new window with "target="\_blank" <a href="link" target="\_blank"> Open this link</a> (opens in new window)
* Link to id's on a page using hashtag <a href="#top">Top</a> (if you have an id=”Top”)
* Combine different pages with id links <a href>="../index.html/#contact">Contact</a>
* Inline frames or iframes are little windows in a page that show another page
  + Use <iframe src=”url” width=”#” height=”#”></iframe>
  + Older html versions use frameborder and scrolling attributes, but html5 does not
  + Html5 uses the seamless=”seamless” attribute where scrollbars are not desired (not supported on older browsers)
  + Useful for adding google maps

**Images**

* Images are coded using <img src="image.location" alt="image description" title="text displayed when hovering over the image" /> (always include the alt and title)
* Inside the <img> element, you can include width="" and height="" to specify px, though this is being done increasingly more with css
* Old html could align the image to a side and have text next to it using align="right/left" though this is removed from html5. do it now using css. also top/middle/bottom
* if wanting a caption for a figure, <figure><img...><br /><figcaption></figcaption></figure> (could also use <p></p> around figcaption instead of <br />)
* Figures can include multiple images, but will display only one caption
* Use gif (or png) format for simple and flat colors, use jpeg for complex pictures
* Save images at the same px dimension as used for the website
* Make an image a link like so <a href=”weblink”><img src=”image.location” alt=”image description” title=”text displayed” />text here would display below the image</a>

**Tables**

* Tables are setup using <table></table> and add data one row <tr></tr> at a time by entering data in each cell <td></td> from left to right
* Three sections of a table headings <thead></thead> body <tbody></tbody> and footer <tfoot></tfoot>
* You can add table headings <th></th> inside a <thead></thead> section
* Blank cells should have <th></th> or <td></td> as fillers
* You can specify "row" or column "col" title with scope <th scope="col"></th>
* You can have data/headings span multiple columns or rows <td colspan="4"></td> or <td rowspan="2"></td>

**Forms**

* Form controls live in the <form></form> element
* <form></form> forms must have an action, and should have method & id attributes
* <form id=”form1” action=”http://www.webpagetoreceiveformdata.com/forminfo.php” method=”get”>

<p>Form controls here

</p>

</form>

* The “get” method adds values to the end of the url specified
  + Use for short forms (search boxes)
  + Just retrieving info and not trying to add/delete info from a database
* The “post” method sends values in http headers
  + Use to allow file upload
  + For long forms
  + Use for sensitive data (passwords)
  + Trying to add/delete info from a database
* A “size” attribute was used in older code to create a box to enter info (use css now)
* <input /> is used to create a type of input and is self closing (just add /> at the end)
  + Inputs need a type & name
    - Type=”type of input”
    - Name=”name of the piece of data”
    - Maxlength=”max # of characters to accept”
    - Checked=”checked” will default a radio or checkbox as already checked
    - value=”” to supply values for hidden or default text fields
  + The “password” type will hide characters, but does not mean secure data
* For full security, you should use Secure Sockets Layer (SSL)
* The <textarea></textarea> element is used to create a multiline input
  + Probably want to use the POST method so url sent to the server isn’t huge
  + Text between the opening/closing tags will be displayed in the text box
  + This text will not be deleted automatically unless using JavaScript
  + Older code uses cols=”” and rows=”” to specify the size of the box (css is used now)
* Hidden inputs type=”hidden” use to pass information to the source file, like setting a value to a variable (action = “add\_new\_data”) would set the value of an action variable (must include a value)
* Text inputs type=”text”
  + Setting value=”” will have a default value in the text box
* Number inputs type=”number” requires a number (validation)
  + Min=”” and max=”” attributes can specify a range
* Password inputs type=”password” will hide characters as they are typed
* Radio inputs type=”radio” can select just one choice & can set default checked=”checked”
  + Must give all of your radio choices the same name=”” so only one can be selected
  + Give them different value=”” for your different options
* Checkbox inputs type=”checkbox” can select multiple
  + value=”” supplies the value for a checkbox (not needed if all are different names)
  + name=”” supplies the key for a checkbox
    - Either make all the names unique if they are not related or…
    - If using names to store checkbox info in an array, related checkboxes should have the same name and the name=”” should have brackets to store data in an array (name=”jobs[]”)
* Dropdown lists are created using <select></select>
  + <select name=”field\_name”></select> with the <option></option> provided in the middle
  + Dropdown lists include <option value=”option\_name”>Option Name</option>
  + Can use php (or probably javascript) to populate a list box (see PHP library)
  + Can include the selected=”selected” attribute for an option if you want it preselected
  + Can use the size=”#” attribute in the <select> element to show more than 1 option
  + Can use the multiple=”multiple” attribute in the <select> element to allow users to click more than one option. Let them know that they should hold control on a PC and command on a Mac to do this.
    - The name=”” for this should include [] to store in an array (name=”field\_name[]”)
* File uploads are allowed using the type=”file” and you must use method=”post”
* Create a submit button with type=”submit”. Be sure to include value=”Words on the submit button”.
* Your submit button can be an image with type=”image” src=”imageurl”
* Type <button></button> allows you to combine images and text on a button using <img src=”” alt=”” />
* You can also collect hidden info with the type=”hidden” that also needs a name and value.
* The type=”date” attribute is newer and not supported by all browsers, and is treated as text by those.
* The type=”email” is also newer but good for collecting email addresses. Will be text box if not supported. Will require a valid email address. Type of form validation.
* The type=”url” is treated similarly to “email” and is a type of validation
* The type=”tel” will do a telephone number (but does not do any validation) can add placeholder=”999-999-9999” to the input as an input mask
* The type=”search” will add a search box
* The placeholder=”Placeholder text” attribute will add text that disappears in boxes, but does not work with textareas.
* It helps to give labels <label></label> to inputs. Two ways:
  + <label>Age: <input type=”text” name=”age” /></label>
  + <input id=”female” type=”radio” name=”gender” value=”f” />

<label for=”female”>Female</label>

* You can use the <fieldset></fieldset> element around related controls. This will box them off.
  + The <legend></legend> element will allow you to name the fieldset with the name appearing in the box
* You can add form validation using HTML5, though some browsers may not support it, and it’s often done using JavaScript
  + The attribute required=”required” is often added

**Video & Audio**

* See page 211 of HTML & CSS for info on posting videos and audio to a web page

**CSS STYLING**

**Basic Layout**

* There are no headings or anything. Just start writing using selectors { } and save as .css
* Choose the element(s) or items to style using selector(s), which can be separated by commas
* Set the style using declarations, which have properties and values separated by colons
* End each declaration with a semicolon
* Selector1, selector2, selector3 {property: value; property value; property: value;}
* Selector.class { } will choose only those selectors of that class (ex. p.intro selects all <p> with class=”intro”
* Css can be coded using a separated file (most common), listed at the beginning of a page, or embedded in html
* You can create comments in css using /\* type your comments here \*/

**Linking to css File**

* Best method with website with more than one page
  + Can reference one stylesheet for multiple pages
  + Keeps content and styling separate
  + Can change the styles for all pages by just altering one css file
* Provide the <link href=”path/tofile/location.css” type=”text/css” rel=”stylesheet” /> in the <head></head> of html pages
* Three attributes are needed:
  + href=”path/tofile/location.css”
  + type=”text/css” (should have this value for css files)
  + rel=”stylesheet” (relationship to linked file should be stylesheet for css)
* It is possible to link more than one stylesheet to an html file
  + Method 1: link one stylesheet using html, then import other stylesheets from that one linked stylesheet
    - Your first lines on the linked stylesheet should use @import url(“filename.css”);
    - You can add several of these, but do so before any styling code
    - Rules in the @import pages will be overruled by the main file
  + Method 2: add multiple <link rel=”stylesheet” type=”text/css” href=”location/filename.css” /> links in the <head> of your html code, one for each stylesheet
    - Rules in the later links take precedence over stylesheets linked firsts

**Using css Styles in an HTML Page**

* css styles are contained within the <style></style> element usually within the <head></head> of an html doc
* Give the style the css attribute <style type=”text/css”></style>
* Keep an eye out for older pages that provide the style attribute within an html element (old practice)

**Cascading css Rules for Precedence**

* Last Rule 🡪 the selector/style that occurs last will take precedence
* Specificity 🡪 h1 is more specific than \*, and id’s are more specific than element types
* Important 🡪 you can add !important before the semicolon to override everything else
  + Selector {property: value !important;}

**Inheritance**

* Properties like font-family and color will apply to everything within an element like <body></body>
  + All <p></p> or <ul></ul> and such would have this style
* Properties like background-color and border apply to the entire section and not every child element
* You can add a value of inherit to a property to force a child to inherit a value #title {padding: inherit;}

**Table of most common css Selectors**

|  |  |  |
| --- | --- | --- |
| **Selector** | **Meaning** | **Example** |
| Universal selector | Selects everything in the html doc | \* { } |
| Type selector | Selects certain element types | h1, h2, p { } every occurrence |
| Class selector | Selects everything in that class  Use with class=”classname”  Can specify types/classes together | .classname { } or  p.classname { } only paragraphs of that class for example |
| ID selector | Selects a unique id attribute  Use with id=”idname” | #idname { } |
| Child selector | Selects elements that are direct children of another element | li>p { } selects any <p> elements directly below <li> element but not other <p> elements |
| Descendant selector | Selects descendants that may not be direct children | p a { } selects any <a> elements inside a <p> element even if nested |
| Adjacent sibling selector | Selects the first occurrence of an element after a specified element | h1+p { } would select the first <p> element immediately after an <h1> element but not other <p> elements |
| General sibling selector | Selects all occurrences of an element that are siblings of a specified element | h1~p { } would select both sibling <p> elements next to the <h1> element |

**Table of css Selectors that Have an Attribute with a Specific Value**

|  |  |  |
| --- | --- | --- |
| **Selector** | **Meaning** | **Example** |
| Existence | Matches every element/attribute combo (value doesn’t matter) | p[class] will select every <p> element that has a class attribute |
| Equality | Matches every element, attribute, and value specified | p[class=”dog”] will select any <p> element that has class=”dog” |
| Space | Matches every element, attribute, and value that includes that word somewhere | p[class~=”dog”] will select any <p> element that has class=”dog in here somewhere” |
| Prefix | Matches every attribute whose value begins with something | p[attr^”d”] will select any <p> element with an attribute that has a value that begins with “d” |
| Substring | Matches any attribute within the element specified that contains the specified letters somewhere | p[attr\*”do”] will select any <p> element with an attribute that has a value with the letters “do” in it somewhere |
| Suffix | Matches any attribute within the element specified that ends in the letters specified | p[attr$”g”] selects any <p> element with an attribute that ends in the letter “d” |

**Testing css Styles in Multiple Browsers**

* Websites can show you what your page will look like in multiple browsers
  + Browsercam.com
  + Browserlab.adobe.com
  + Browsershots.org
  + Crossbrowsertesting.com
* Test on PC and Mac
* Use sites to try and debug if unable to fix issues
  + Positioniseverything.net
  + Quirksmode.org

**Colors in css**

* Three ways to specify color
  + RGB 🡪 color: rgb(#, #, #); with numbers from 0-255
  + Hex codes 🡪 color: #af0ed8; with each character ranging 0-9 then a-f
  + Color names 🡪 color: DarkCyan;
* Go to <http://paletton.com/#uid=73o0u0kllllaFw0g0qFqFg0w0aF> for color schemes
* You can add the opacity property to colors with values of 0.00-1.00 in terms of percentage
  + Ex. opacity: 0.15; means an opacity of 15%
  + Adding this as a property will influence child elements
* You can also do opacity by using rgba instead of rgb 🡪 color: rgba(0, 0, 0, 0.5); means 0 red, 0 green, 0 blue, & 50% opacity
* Opacity and rgba may not be supported by all browsers
  + Fall back colors use rgb first then rgba on the same selector (rgba would take precedence if supported)
* Newer css3 can use hsl or hsla to do colors, but should probably use rgb or hex first as a fallback
  + hsl describes hue, saturation, and lightness
  + Hue is the color represented as an angle on a color wheel from 0-360 (degrees)
  + Saturation is the amount of gray in a color where 0% is gray and 100% is full color saturation
  + Lightness is the amount of white/black where 0% is black, 50% is normal, and 100% is white
  + Ex. hsl(165, 95%, 55%)
  + hsla is the same except that it adds the alpha or opacity just like rgba hsla(165, 95%, 55%, 0.44)
  + hsl is a good way to use different shades of the same color

**Text**

* The property font-family: ; is used to specify which font to use using selectors
  + You can stack fonts in font-family so that if the first one isn’t supported, it will try the next
  + It is common to add the generic type at the end (i.e. serif, sans-serif, monospace, cursive or fantasy) though cursive or fantasy may do weird things
  + If a font name is more than one word, it should be placed in quotation marks
  + Good practice is not to use more than three different fonts on a page
* The property font-size: ; is used to specify the size of the font
  + Default for most browsers is 16px (and is good practice to use for the body font-size)
  + Can specify pixels font-size: 12px; (probably the best choice)
    - Could use pt instead of px, but only do this on “printer-friendly” versions of pages
  + Can specify percentages font-size: 200%; 🡪 percentages multiply, so if 16px is default, 75% is 12px, then another nested 75% would be 9px (percentages is probably second best choice)
  + Can specify ems (sort of like doing percentages as a decimal 1.5em is 150%, but weird to use)
* Can use @font-face { } to make a computer download the font, but this has licensing issues see p. 277-278
* Formatting text
  + Use text-align: ; to specify left, right, center, or justify
  + Use text-indent: ; to indent text 🡪 usually with px or ems
    - Can use a negative value if you want text to appear in code/search engines without appearing on the page (such as a title if using an image)
  + Use vertical-align: ; to align text in <td> or <th> cells or possibly with <img> <em> or <strong> elements
    - Values: baseline, sub, super, top, text-top, middle, bottom, text-bottom
    - Additional values: length in px or em or %
  + Use font-weight: ; to specify normal or bold (default is normal, can specify normal within a bold section)
  + Use font-style: ; for normal, italic, or oblique (oblique is like normal but titled)
  + Use :first-letter or :first-line to alter the first letter or first line of an element
    - Ex. p.intro:first-line {font-weight: bold;} would bold the first line of the <p> with class=”intro”
  + Use text-transform: ; to change case to uppercase, lowercase, or capitalize (for first letter only)
  + Use text-decoration: ; to decorate 🡪 none, underline, overline, line-through, blink (don’t use blink)
  + Can add word shadows using text-shadow: 0px 0px 0px #000000;
    - The first px is how far left or right the shadow goes
    - The second px is distance to top or bottom of shadow
    - The third value is the amount of blur to the shadow
    - The fourth value is the color
  + Use line-height: ; a good amount is 1.4-1.5em and you should use em rather than px here to account for users changing fonts in their browsers
  + Use letter-spacing: ; to change the amount of space between letters (use em)
  + Use word-spacing: ; to change the amount of space between words (use em)

**Bullet Point/Numbered List Formatting**

* Can set the width, margin, padding, etc. for lists and list items
* list-style-type: ; (changes the shape of the marker)
  + Works on ordered lists <ol>, unordered lists <ul>, and list items <li>
    - Unordered lists can have none, disc (normal), circle (not filled), and square
    - Ordered lists
      * decimal 🡪 1 2 3
      * decimal-leading-zero 🡪 01 02 03
      * lower-alpha 🡪 a b c
      * upper-alpha 🡪 A B C
      * lower-roman 🡪 i ii iii
      * upper-roman 🡪 I II III
* list-style-image: url(“image/location.png”); (uses an image as the bullet point)
  + Use on <ul> or <li> elements
* list-style-position: ; outside sets bullets outside the box, and inside sets the bullets inside that box
* list-style: ; lets you set the style type, image, and position properties in any order

**Table Properties**

* Good policy to distinguish <th> table headings with text-color background-color or uppercase
* Helps to maybe shade alternate rows
* Use text-align: right; to align numbers to the right of cells
* width: ; 🡪 table width
* padding: ; 🡪 space between cell border and cell content
  + Adding padding helps readability
* text-transform: ; 🡪 set to uppercase, lowercase, or capitalize
* letter-spacing: ;
* font-size: ;
* border-top: ;
* border-bottom: ;
* text-align: ;
* background-color: ;
* :hover {} (to change background or text color)
* empty-cells: ; can show, hide, or inherit the rules specified to the table
* border-collapse: collapse; will make all borders the width of the border-width: ; that you specify without having adjacent borders combining (empty-cells: ; styling will be ignored and cells shown)
* border-collapse: separate; will make border spacing and empty cell rules apply with cells separated
* border-spacing: 5px; will set 5px between vertical and horizontal cells while border-spacing: 5px 10px; will set 5px between left-right cells and 10px between rows

**Styling Links/Buttons/Fields**

* Use a:link { } to style links that have not yet been visited (with color/underline etc.)
* Use a:visited { } to style links that have been visited
* Use \_\_:hover { } to change the appearance of elements (buttons/links etc.) when hovering (does not work with devices that only have touchscreens)
  + Need example to display text on hover
* Use \_\_:active { } to style things as they are clicked
* Use \_\_:focus { } to add color to an element when it is being used (like a text input box). This shows the active field.
* The order should go :link, :visited, :hover, :focus, :active

**Styling Forms**

* Below items are added within input {} elements unless otherwise specified
  + font-size: ;
  + color: ; (text color)
  + background-color: ;
  + border: ;
  + border-radius: ;
  + input:focus, input:hover {} can be used together like this to set what happens when either in an input field or hovering over it with your mouse
  + background-image: url(“image/url.png”); to add a picture in an input
* Common submit button styling
  + color: ; (text color)
  + text-shadow: ; gives 3D look to the text by adding a shadow
  + border-bottom: ; give 3D feel
  + background-color: ; to make it stand out (can use gradients)
  + :hover {} (#id:hover or ex. input:hover or input#id:hover)
  + text-align: right;
* Common Fieldset/Legend Styling
  + width: ;
  + color: ; (text color)
  + background-color: ;
  + border: ;
  + border-radius: ;
  + padding: ;
* Align the edge of input fields by making their <labels> have the same width and set float: left; and text-align: right; also add some padding-right: ;
* cursor: ; (most often used with :hover {}
  + auto
  + crosshair
  + default
  + pointer
  + move
  + text
  + wait
  + help
  + url(“cursor.gif”);

**Boxes**

* You can specify the height and width of boxes in px, %, or ems (px most popular, % or ems may work a little better across different screen sizes)
* If another element like <p> is within a box (like a <div> element) then you can specify height and width of the inner element as a percentage of the larger box
* Use min-width and max-width to specify the largest and smallest sizes your box should take
  + Sizes should change if you adjust the window size of the browser or screen size
* Use min-height and max-height the same way as with width
* You should specify what to do if the text doesn’t fit with the overflow attribute
  + overflow: hidden; will hide any text that doesn’t fit
  + overflow: scroll; will add scrollbars to any box with overflow text
* Use margin: ; to add space between boxes
  + Usually use px
  + Also margin-top, margin-right, margin-bottom, margin-left
  + Also margin: 1px, 2px, 3px, 4px; for top right bottom left
  + Also margin: 20px, 10px; for 20px top and bottom, and 10px right and left
* Use padding: ; to add space between content within a box and its border
  + Usually use px
  + Padding makes a box bigger
  + Also padding-top, padding-right, padding-bottom, padding-left
  + Also padding: 1px, 2px, 3px, 4px; for top, right, bottom, left
  + Also padding: 5px, 10px; for 5px top and bottom, and 10px right and left
* Use background-color: ; to change the background color.
* Use border-width: ; as expected. Specify with px or with thin, medium, or thick
  + Also border-top-width, border-right-width, border-bottom-width, border-left-width
  + Also border-width: 1px, 2px, 3px, 4px; so top is 1, right is 2, bottom is 3, and left is 4
* Specify a border-style: ;
  + Solid, dotted, dashed, groove, ridge, inset, outset, hidden/none
  + Also border-top-style, border-right-style, border-bottom-style, border-left-style
* Specify a border-color: ;
  + Use rgb, hex, rgba, hsl, or hsla (or color names)
  + Also border-top-color, border-right-color, border-bottom-color, and border-left-color
  + Also border-color: hexfortop, hexforright, hexforbottom, hexforleft;
* Use border-radius: ; to create rounded edges on a box
  + Can use px or % or em
  + Can specify border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius, border-top-left-radius or use four values 🡪 border-radius: 2px, 5px, 2px, 5px;
  + Can create shapes by specifying different width/height values
    - border-top-left-radius: 80px 50px; will have 80px width and 50px height
    - Using four values separated by a / will do the four horizontal then the four vertical
      * Border-radius: 1em 4em 1em 4em / 3em 2em 3em 2em; will have top-right be 1em horizontal and 3em vertical, bottom-right be 4em horizontal and 2em vertical etc.
  + Make a circle by starting with a square box and making the border radius equal to the px length of a side or using 50%
  + Use -moz-border-radius: ; and -webkit-border-radius: ; to help older browsers
* Use box-shadow: ; to add shadows to boxes
  + Uses the following info in this order:
    - Horizontal offset (required)
    - Vertical offset (required)
    - Blur distance (if omitted, the shadow is a solid line)
    - Spread (positive value makes it expand, negative value makes it contract)
    - Color (required)
    - Inset can be added before any values to make an inner shadow (within the box edges)
    - Ex. box-shadow: inset 2px 3px 4px 5px #777777;
  + Use -moz-box-shadow: ; then -webkit-box-shadow: ; then box-shadow: ; to help older browsers
* The border: ; property allows you to specify the width, style, and color of a border in that order
  + Ex. border: 2px, ridge, #23e5a7;
* Centering a box on a page
  + You must first specify a width, because otherwise the box will be the page width (or max width within the element in which it’s contained)
  + Set the left-margin: auto; and right-margin: auto; so the page will make them equal
  + In older browsers, the parent element must have text-align: center; which will then be inherited by the child box
    - You should then use text-align: left; or whatever you want in the child box to adjust
* Display 🡪 used to make inline elements appear as block and block appear as inline
  + Often used with <ul> and <li> to make site navigation
  + Display: inline; will make block-level appear as inline (like site navigation)
  + Display: inline-block; makes a block-level flow like an inline, while retaining block-level features
    - Use to center an unordered list (combine with text-align: left;)
  + Display: none; will hide an element on the page from everyone including screen readers
* Visibility 🡪 will hide boxes but leave a space for them
  + Visibility: hidden; hides the element from everyone (including screen readers) leaving a space
  + Visibility: visible; shows the element
* Screen reader only 🡪 define a class (class=”sr-only”) and then position the item left: -10000px; top: auto;, and this box cannot have 0 height/width, so 1px works well for both, also set overflow: hidden;
  + Sr-only text is useful when conveying info from graphs/charts to visually impaired
* Use border-image: url(“images/imageurl.jpg”); to make an image the border of a box
  + Need three pieces of info:
    - Image url
    - Where to slice the image
    - What to do with straight edges (stretch, repeat, round🡪scales image so tiles fit)
    - Ex. border-image: url(“images/image1.gif”) 11 11 11 11 round;
  + Need to specify border width so the image is shown
  + The image will be sliced into nine pieces (think tic-tac-toe style)
    - The corners are the corners
    - The sides can be repeated or stretched
  + Might need to repeat the code three times using -moz-border-image: ; then -webkit-border-image: ; then border-image: ; to help oder versions of browsers

**Responsive styles that adapt to screen size**

* Use media queries to specify rules to apply based on viewing screen
  + @media (max-height: 1920px) { p {text-size: 10px;} } is an example that would make paragraphs have 10px text on screens 1920px height or less
  + Can also do media queries based on width
* Make images fit on small screens by making them block elements with height: auto;
  + img { max-width: 100%; display: block; height: auto; } will not allow images to appear larger than original, display them as block elements, and auto adjust the height based on the size of the block (screen)
* Format images for “retina” screens by styling the width and height as 50% of the original px value
* Use relative dimensions for elements using…
  + width: 40vw; (element is 40% of viewport’s width)
  + height: 10vh; (element is 10% of the viewport’s height)
  + vmin: 75vmin; (element is 75% of the viewport’s shorter dimension)
  + vmax: 65vmax; (elementis 65% of the viewport’s larger dimension)
    - Can use these to specify width, height, font-size, max-height, min-height, for parameters and boxes etc.

**Page Layout**

* Positioning Schemes
  + Normal flow (all block level elements will appear on new lines regardless of width: default action)
    - Position: static; (this is the default setting and does not need to be coded)
  + Relative positioning
    - Position: relative; will allow you to then add top: ; bottom: ; left: ; and right: ; values to offset the object from where it would appear in normal flow (use px % or em)
  + Absolute positioning (remain fixed and seem to take up no space in relation to other elements)
    - Position: absolute; sets where an item is (and other elements can overlap)
    - Use top: ; bottom: ; left: ; and right: ; to set the position
    - You may need to adjust the width/height of this object and other objects to not overlap
  + Fixed positioning (type of absolute positioning where the element doesn’t move even when scrolling)
    - Position: fixed; will keep an item in its position on the screen even when scrolling
    - Use top: ; bottom: ; left: ; and right: ; to set the position
    - Add a margin to the next element so it is visible if setting a fixed object at the top of the window
  + Floating elements (turns a block level element into kind of an inline style with other elements flowing around it)
    - Float lets you set an element to the far right or far left of the containing element
    - Everything else in the containing element will flow around it
    - Always add width: ; when using position: float;
    - Good for quotes, pictures, and things to stand out in a paragraph
    - You can add the clear: ; property to specify that nothing should touch the left, right, or both sides of a floated element (use if things aren’t spaced correctly)
    - If the border of a floated container isn’t working right, set overflow: auto; and width: 100%;
    - Multi-column layouts (see 960 grid section)
      * Use <div> elements and add classes of “column1” “column2” etc. to help
      * Each div can have other elements in it
      * Set the width of each column
      * Set float: left;
      * Add a margin: ; to add space between the columns
      * Can style all classes the same .column1of3, .column2of3, .column3of3 { }
  + z-index lets you say which box appears on top if boxes overlap
    - The z-index: ; property is a number that says how close an object sits to the front: the higher the number the more on top an object is

**Images**

* Can apply these to the <img> or <figure> elements
* Control height and width just like any other box
* Good practice to have general sizes as classes (i.e. img.large or img.small etc.)
* Can use the float property on “align-left” or “align-right” classes in conjunction with a margin to ensure text doesn’t touch the image
* To center an image in the middle of the screen (with no text around it) you need to set the display: block; with margin-left and margin-right set to auto
* **Background Images**
  + Will repeat to fill the box they are in, use background-image: url(“directory/filename.jpg”);
  + Complex or large files can make pages take longer to load
  + Use background-repeat: ; in conjunction with background-image: ;
    - : repeat; is the default and will repeat the image in the x and y directions
    - : repeat-x; will have the image only repeat in the x direction
    - : repeat-y; will have the image only repeat in the y direction
    - : no-repeat; the image is only shown once
  + Use background-attachment: ; in conjunction with background-image: ;
    - : fixed; the image stays in the same position on the page
    - : scroll; the image moves as the user scrolls through the page
  + Use background-position: ; to specify where the image appears if not repeated
    - Left top; left center; left bottom; center top; center center; center bottom; right top; right center; right bottom
    - Could also specify a pair of px or % with the horizontal specified first
  + Use background: ; by itself to specify these properties in this order:
    - Color, image, repeat, attachment, position (can skip any you don’t want)
  + Rollovers and sprites
    - An image sprite is a single image that contains multiple views of a button
    - You can use :hover and :active to change the background-position of the image so that the user sees the button you want them to (i.e. the image file contains three version of a submit button (at different positions) as you :hover or :active a button, the position changes so they see the right one
    - The browser only has to load one image, and this can save time loading a page versus loading different images if you want to use a picture for a button

**Screen Sizes**

* Good practice to design pages no more than 960-1000px wide to display on most devices
* Try to get most important info in top 570-600px so it is viewed without scrolling
* Fixed width layouts can work well for desktop or laptop screens, and tend to be more vertical (tend to use px)
* Liquid layouts will contract to fit the width of the screen, and adapts better to modifications by users to zoom (tend to use %)
* **Fixed Width Pages:**
  + Set the width of items in px (max page width about 960 px)
    - Can have an extra <div> or <page> element to have a background image that takes up 100% of the screen
  + Center content on the screen by setting margin-left and margin-right to auto
  + Use px for padding/margins too
* **Liquid Layout Pages:**
  + Set the width of items in % (good idea to use only about 90%)
  + Use % for margins
  + Use em for padding
  + Use max-width to keep things from stretching too much
  + Use min-width to keep things from getting squashed too much

**Flexbox (sometimes it seems results are unpredictable)**

* You can set the display: flex; for an element to alter its layout (required for other flexbox code to work)
* You can also specify the flex-direction: ; once you have set the display to flex
  + Directions include row, column, row-reverse
* The main axis of the direction is a line (horizontal for flex-direction: row; and vertical for column;)
* The cross axis is perpendicular to the main axis
* Aligning objects in a flex container when they don’t fill it completely (all examples start with justify-content: ;)
  + center; to center it
  + flex-start; aligns row to the left and column to the top
  + flex-end; aligns row to the right and column to the bottom
  + space-between; will align objects to the outer edges of the main axis with a space between them
  + space-around; will put space around all sides of the objects
* The align-items: ; tag is used to align items in the cross axis (vertically for rows and horizontally for columns)
  + flex-start; aligns to the top of rows and to the left of columns
  + flex-end; aligns to the bottom of rows and to the right of columns
  + center; aligns to the vertical center of rows and to the horizontal center of columns
  + stretch; will stretch things top to bottom in a row and right to left in a column
  + baseline; aligns items to their baselines (try it to see what happens)
* The flex-wrap: ; property will tell css to wrap items
  + nowrap; is the default setting and will not wrap anything
  + wrap; will wrap left to right in a row and top to bottom in a column
  + wrap-reverse; is the same as wrap but backwards

**CSS 960 Grid**

* The 960 grid system uses 12 columns of 60px wide each
* There are 10px margin on each column, meaning 20px between each column and 10px left/right margins
* I downloaded some code in the D drive that can be linked or copied
* To code from scratch (also see page layout section for help)
  + Use classes to set the main container to 960px
  + Set classes for elements to take up the appropriate number of columns grid\_4 for 4 columns etc.
    - .grid\_1 should be 60px, but each successive grid should add 80px (20 for margins) (i.e. 60, 140, 220, 300, 380, up through 940px
    - All grids should display: inline; float: left; margin-left: 10px; margin-right: 10px;
    - There are some additional codes to help with other things if you have issues in the downloaded stylesheet
    - You may need to add clear: left; (or right) to fix any issues with row height
      * Clear: left; will not allow a child object’s left side to touch another child object within the same parent object (i.e. two different <p> within the same <div>)
      * Only use the clear property on the one child object that is out of place

**CSS Grid (older version) (sort of like using a table layout)**

* You need a section to make a grid like a <div></div> and this will be your container
  + Any elements inside (like other <div></div>’s) will become grid items
  + Convention says to give each item a class naming the item (<div class=”item1”>Grid Title</div>)
* You must set the display: grid; to use CSS Grid features in the container class
* This will apply to the element you style (parent element/container) and all of its children elements (items)
* Convention is easier to style if parent element has class=”container” and each child element has class=”item1”…
* Each item within the parent container will be placed in the grid
* px or em can be used for sizing
* Can also use the minmax(#, #) function to specify the minimum and maximum sizes of a row/column
  + grid-template-columns: ; will let you set the number of columns and their width
    - grid-template-columns: 50px 50px 50px 50px; will create 4 columns each with 50px width
    - can use repeat to make life simpler
      * grid-template-columns: repeat(100, 50px); will make 100 columns each 50px wide
      * grid-template-columns: repeat(2, 1fr 40px) 60px; will make 5 columns with the following widths: 1fr 40px 1fr 40px 60px
  + grid-template-rows: ; will set the number of rows and their height
    - grid-template-rows: 100px 100px; will produce two rows each of 100px height
    - can use repeat to make life simpler
      * grid-template-rows: repeat(200, 40px); will make 200 rows each 40px height
      * grid-template-rows: repeat(3, 2fr 30px) 50px; will make 7 columns with the following widths: 2fr 30px 2fr 30px 2fr 30px 50px
      * Repeat also comes with auto-fill option
        + auto-fill will fill as many rows/columns as will fit within your container
        + Combine auto-fill with minmax to create flexible layouts

Ex. repeat(auto-fill, minmax(30px, 100px)

* + - * Repeat also comes with auto-fit option
        + auto-fit will resize cells to fit the container

If minmax sizes are exceeded, cells will be added or moved to another line accordingly

* + fr used as units sets the column or row to a fraction of the available space
  + auto sets the column/row to the width/height automatically
  + use % to set the percentage of the container
  + Can combine units of px, fr, auto, em, or % in a grid-template… style
* Add a gap between columns/rows using the grid-column-gap: ; or row using grid-row-gap: ;
  + Use px or other units to specify the size of the gap between each column/row
* Or do both using grid-gap: ;
  + The first number specified will set the row gap the second will set the column gap
  + One number would be used for rows and columns
* Make items span more than one column of the grid using grid-column: # / #;
  + Think of the #’s used for this css code as the grid lines
  + A 3x3 grid (3 columns 3 rows) would have 4 lines going across and 4 lines going down
  + grid-column: 2 / 4; would span the second and third columns of a 3x3 square grid
* The same works with grid-row: # / #;
  + grid-row: 4 / 6 would span the 4th and 5th rows of a 5 row (or more) grid
* Each “cell” of the grid can have a justify-self: ; property to align contents horizontally
  + Default is justify-self: stretch; which will fill width of the cell
  + justify-self: start; aligns the content to the left of the cell
  + justify-self: center; aligns content to the center of the cell
  + justify-self: end; aligns content to the right of the cell
* Each “cell” can also align vertically using align-self: ;
  + align-self: start; aligns content to the top of the cell
  + align-self: center; aligns content to the middle of the cell
  + align-self: end; aligns content to the bottom of the cell
* You can align/justify all items at once using
  + justify-items: ; for aligning horizontally
  + align-items: ; for aligning vertically
* Grid areas (defining each cell)
  + Use grid-template-areas: to layout the “name” of each cell
  + grid-template-areas:

“header header header”

“advert content content”

“footer footer footer”;

Would use a 3x3 grid where the first row is one area called the header, the second row has two areas called advert and content, and the third row is an area called the footer

* use a period . to designate an empty cell in grid-template-areas
* use grid-area: ; to specify where a class can go, referencing the template area that was defined
* use grid-area: ; to define where an object is using four numbers
  + .item1 { grid-area: 1/1/2/4; } would set any object with the class=”item1” to…
    - 1 = horizontal line to start at
    - 1 = vertical line to start at
    - 2 = horizontal line to end at
    - 4 = vertical line to end at
* Grid within a grid
  + You can edit the class for one of your grid elements into a grid
    - Ex. .item1 { display: grid; }
    - Then the element with class=”item1” will act as a container with elements nested inside acting as items