**CSS** - Cascading Style Sheets

1. **Intro**
   1. Describes how HTML elements are to be displayed
   2. Helps split structure and presentation
   3. Without CSS, it would be difficult to change themes/color pallets of websites quickly
   4. CSS Syntax: selector { property: value; }
   5. Browsers come with a default stylesheet
      1. To override these, you will need to add CSS to zeroize the default
         1. Ex (removes underlining of links): a { text-decoration: none; }
      2. Default box-sizing is not ‘border-box’
         1. This means on default stylesheets, width is actually border + padding + content
         2. To make width include border + padding + content:
            1. body { box-sizing: border-box; }
   6. CSS framework ‘Bootstrap’ is commonly used by developers, and can be found here:
      1. <http://getbootstrap.com/css/>
   7. Media Queries allow developers to specify blocks of CSS that are only activated at certain screen widths
      1. They typically go at the bottom of your CSS file
      2. Ex: /\* up to iPad-Portrait width \*/

@media only screen and (max-width:768px)

{ Insert CSS code here for small screens}

1. **Declaring variables to recall as values**
   1. New feature, you can do it, look up how and fill this out
2. **Selectors**
   1. Tag: p{ CSS }
   2. Universal: \*{ CSS }
   3. Class .className { CSS }
   4. ID: #idName { CSS }
      1. Case Sensitive!
   5. Attributes: [type=”password:] { CSS } and [name] { CSS }
   6. Pseudo Class: input:focus { CSS }
      1. The most common pseudo-classes target the state of links (<a> tags)
         1. a:link – style links that haven’t been visited yet
         2. a:visited – style links that have been visited
         3. a:hover – style links (or any element) as the visitor hovers their mouse over it
         4. a:active – style the link when the user clicks (but before releasing)
      2. Another useful focus is ‘first-of-type’ and ‘last-of-type’
         1. p:first-of-type { CSS }
            1. This only edits the first <p> element
   7. Pseudo Element: blockquote::before { CSS }
   8. Combined: reduces scope
      1. Syntax: (selector1)(selector2) { CSS }
      2. Ex: p.intro { color:yellow; }
      3. This is the same as & functionality, i.e. only elements with both selector 1 (<p>) and selector 2 (class=”intro”) will be modified
   9. Groups: increases scope
      1. Syntax: (selector1), (selector2) { CSS }
      2. Ex: h1, .special { color: red; }
   10. Descendants:
       1. Styling a parent will also style all descendants
          1. ex: <section> <h1>Words</h1> <p>More Words</p> </section>
          2. Styling <section> will also style both <h1> and <p> elements
       2. If you separate selector names with a space, the second selector targets descendants of the first
          1. ex: section h1 { color:red }
          2. This will target all <h1> elements in <section>
       3. If you separate selector names with a >, only immediate children are targeted
          1. ex: <body> <h1>Child></h1> <div><h1>GrandChild</h1></div> </body>
          2. ex: body > h1 {color: red; }
          3. Only the first <h1> will be targeted, the second <h1> is a grandchild
       4. You can combine Descendant Selectors with Group and Combined Selectors
          1. ex: p.intro a {color:yellow;}
          2. This will only target <a> elements inside <p> elements that also contain class=”intro”
3. **Implementation**
   1. **Inline**
      1. By using the style attribute in HTML elements
      2. Ex: <h1 style=color:blue;”>This is a Blue Heading</h1>
   2. **Internal**
      1. By using a <style> element in the <head> section
      2. Ex: <!DOCTYPE html>

<html>

<head>

<style>

.nice {font-family: sans-serif;}

</style>

</head>

<body>

<h1> This is a blue heading </h1>

<p class=”nice”> This is a red paragraph </p>

</body>

</html>

* + 1. Ex: <!DOCTYPE html>

<html>

<head>

<style>

body {background-color:powderblue;}

h1 {color:blue;}

p {color:red}

</style>

</head>

<body>

<h1> This is a blue heading </h1>

<p> This is a red paragraph </p>

</body>

</html>

* 1. **External** 
     1. By using an external CSS file, this is the preferred method
     2. Can change the look of an entire website by changing one file
     3. Link resides in the <head> section of HTML code, file name ends with ‘.css’
     4. Two primary methods to link to a CSS file
        1. Method 1: <link rel=”stylesheet” type=”text/css” href=”css/global.css>
        2. Method 2: <style type=”text/css”> @import url(css/global.css) ; <style>
     5. When linking to multiple external style sheets, the sheet lowest in the code trumps those above
     6. Ex: <!DOCTYPE html>

<html>

<head>

<link rel=”stylesheet” href=”styles.css”>

</head>

<body>

<h1> This is a heading </h1>

<p> This is a paragraph </p>

</body>

</html>

* + 1. External style sheets can be written in any text editor
       1. Must not contain any HTML code
       2. Must be saved with a .css extension
       3. Ex of .css file:

body {background-color: powderblue;}

h1 {color: blue;}

p {color:red;}

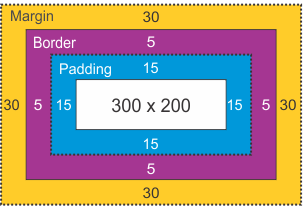
* + 1. External stylesheets can be referenced with a full URL or path relative to the current page
       1. Ex (full path): <link rel=”stylesheet” href=”https://www.w3schools.com/html/styles.css>
       2. Ex (relative path): <link rel=”stylesheet” href=”/html/styles.cdd”>
       3. Ex (same folder): <link rel=”stylesheet” href=”styles.css”>
    2. Use the id attribute to define styles on a specific element
       1. Ex: <p id=”p01”> I am different </p>
       2. In CSS file: #p01 {color: blue;}
    3. Use the class attribute to define a style for multiple specific elements
       1. Ex: <p class=”error”> I am different </p>
       2. In CSS file: #p.error {color:red;}

1. **Cascading and Inheritance**
   1. Nearest Ancestor Wins
      1. ex: <div><p>What color?</p></div>
      2. <p> CSS will trump <div> CSS in the above example
   2. Specificity Rules – highest points wins!
      1. Inline Style – 1,000 pts
      2. ID Selector – 100 pts
      3. Class Selector – 10 pts
      4. Tag Selector – 1 pt
   3. Breaking Ties - when two styles are different and the point values are equal, the one last in code wins
   4. !important trumps all
      1. Syntax: a { color: green !important; }
2. **Layouts**
   1. **Length Units**
      1. Three primary measurements of length:
         1. px – pixel – not exact as some devices have smaller and denser pixels than others
         2. em (“M”) – a measurement based on font size, also not exact, good for human readability
         3. rem (“root”) – generates a length relative to the font-size of the entire page (the html, aka “root” element)
            1. More appropriate for placing components relatively onto a page (according to the users default font preference)
   2. **The Box Model**
      1. Note - use property ‘outline’ to troubleshoot borders as it has no dimensions
      2. Default ‘box-sizing’
         1. Ex: div { width: 300px;

padding:15px;

border: 5px;

margin: 30px; }



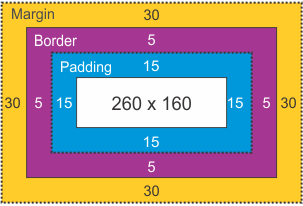
* + - 1. Total Width = width + padding-left + padding-right + border-left + border-right
         1. In the above example, the Total Width is 340px
    1. ‘box-sizing: border-box;’ changes the default box model, making it easier to calculate
       1. Ex: div{ width: 300px;

padding:15px;

border: 5px;

margin: 30px

box-sizing: border-box; }



* + - 1. Total Width = Width (300 px in example)
  1. Properties
     1. The ‘display’ property has 4 values, and is relevant to box layouts
        1. none; - hides the element
        2. inline; - displays the element inline (without line breaks)
           1. Ex of default elements with inline: <em>, <span>
        3. block; - displays the element on a new line (not inline)
           1. Ex of default elements with block: <div>, <section>, <ul>
        4. inline-block; - displays elements inline with modifiable width, height, margin, and padding
           1. By default, inline-block elements have a small space between them
           2. To fix this, assign the property font-size: 0; to the parent element and set the property font-size: 1rem; to the child element
     2. Property ‘margin: 0 auto;’ will center a block element within a container after it is given a width property
     3. Property ‘text-align: center;’ will center an inline element when given to the parent element
     4. Margin
        1. Values can be set per each side, ex: ‘margin-top: 10px;’ (also -right, -bottom, -left)
           1. Alternate Syntax: margin: <top> <right> <bottom> <left>;
           2. Ex: margin: 0 10em 0 10em;
           3. Alternate Syntax 2: margin: <top & bottom> <left & right>;
           4. Ex2: margin: 0 10em;
        2. Values can be set for all sides, ex: ‘margin: 10px;’
        3. Margins ‘Collapse’ - if two adjacent elements share a margin, the margin will overlap
           1. Ex1: p.one { margin-bottom: 1em; }

p.two {margin-top: 1em; }

* + - * 1. The total margin between p.one and p.two is 1em, not 2! They overlap.
        2. Ex2: p.three { margin-bottom: 2em; }

p.four { margin-top: 3em; }

* + - * 1. The total margin between p.three and p.four is 3em.
    1. Position
       1. ‘position: static;’ is the default positioning of all elements on the web
          1. Rendered in the same order they appear in HTML markup, also called the ‘normal flow’
       2. ‘position: fixed;’ elements stay in place when the user scrolls
          1. Outside the normal flow
       3. ‘position: relative;’ elements are outside the ‘normal flow’ and can be positioned using right, top, left, and bottom properties (like fixed elements)
          1. Unlike fixed elements, they are offset in relation to the first parent container with a non-static position
          2. For this reason, they are normally applied to the parent element of the element with ‘position: absolute’ applied to it
          3. Also used in parent elements of ‘position: absolute’ elements
       4. ‘position: absolute;’ should not be used to layout your page
          1. Reserved for special cases, such as pasting an element outside of the normal document flow
  1. Flexbox – a system to organize your layout based on containers that display their content on an x/y axis
     1. There are 4 basic components:
        1. Flex Containers – containers wrap around the content being displayed
        2. Flex Items – the items inside of flex containers
        3. Main Axis – if displayed in row, the x-axis; if displayed in columns, the y-axis
        4. Cross Axis – the perpendicular axis to the main axis
     2. Flex Container Properties
        1. ‘display’ enables Flexbox functionality when set to ‘flex;’
        2. ‘flex-direction’ determines what the main axis is going to be
           1. By default, the value is row (x is the main axis)
        3. ‘justify-content’ property is used to move the content horizontally in an element with ‘display: flex;’ property
           1. ‘flex-start’; - aligns left
           2. ‘flex-end’; - aligns right
           3. ‘center’; - aligns center
           4. ‘space-between’; - aligns even space between
           5. ‘space-around’; - aligns even space between and around
        4. ‘align-items’ property is used to move the content vertically in an element with the ‘display: flex’ property
           1. ‘flex-start;’ - aligns top
           2. ‘flex-end’ – aligns bottom
           3. ‘center’ – aligns middle
           4. ‘baseline’ – aligns at the baseline of the container
           5. ‘stretch’ – items are stretched to fit the container

1. **CSS Grid**
   1. Theory
      1. <div>’s can be diced into a grid of Columns and Rows
      2. By making a <div> display: grid, you’ve made its contents grid items
      3. Explicit - you defined it
      4. Implicit - the browser defined it
   2. Measurements
      1. Fractional Unit - syntax: 1fr, 2fr, 3fr, etc.
         1. A Fractional Unit is a division of space based on how much remains
         2. Far easier to use than percentages
         3. Fractional Units do little to height because height is auto-sized based on the contents height
            1. To get around this, declare a container height; now your Fractional Units work properly
      2. Auto - adjusts to column or row to the max size of the largest single content item
      3. ‘vw’ and ‘vh’ stand for ‘vertical width’ and ‘vertical height’
   3. CSS Keys/Values
      1. **‘auto’ keyword** - replaces a ‘value’ to make the width automatically scale to window size
      2. **‘grid-auto-columns’** - allows you to set the size of all implicit columns
      3. **‘grid-auto-flow: (row or column)’** - allows you to choose if extra elements are added as items in columns or rows
      4. **‘grid-auto-rows’** - allows you to set the size of all implicit rows
      5. **‘grid-gap’** - adds spacing between the ‘tracks’, like a margin
      6. **‘grid-template’**
         1. ‘grid-template-columns’ key will spit out as many columns as you pass as the value
            1. Ex: ‘grid-template-columns: 100px, 100px, 100px’ will output three columns each 100 pixels wide
            2. Rows are auto created based on how many items
         2. ‘grid-template-rows’ key will spit out as many rows as you pass the into value
         3. ‘grid-template-columns’ and ‘grid-template-rows’ can be combined
         4. grid-template-areas: “name name name”
            1. Creates named areas in the grid
            2. Separate named area rows with new sets of quotes, ex:

grid-template:areas:

“sidebar-1 content sidebar-2”

“sidebar-1 content sidebar-2”

“footer footer footer”

* + - * 1. Yields (on a 3x3 grid):

|  |  |  |
| --- | --- | --- |
| sidebar-1 | content | sidebar-2 |
| footer | | |

* + - * 1. A dot ( . ) can be used in place of text to have a ‘dead’ space
    1. **repeat(x, y)** - replaces a value that creates x quantity of y value
       1. Ex: grid-template-columns: repeat(5, 100px)
       2. Adding more values to the end will multiply the X by each Y
       3. Can mix and match, ex:

grid-template-columns: 100px repeat(2, 1fr auto) 100px repeat(2, 1fr)

* + - * 1. This outputs the following: 1px, 1fr, auto, 1fr, auto, 100px, 1fr, 1fr
  1. Sizing Grid Items
     1. Spanning lets you tell an item how many columns or rows to take up
        1. Syntax: grid-column: span 2;

grid-row: span 2;

* + 1. If your span goes wider than your explicit columns, then implicit columns will be made
    2. You can specify which grid column or row an item starts and ends at with:
       1. grid-column-start - specifies which track to start at
       2. grid-column-end: specifies which track to end at
       3. Same with grid-row-start and grid-row-end
       4. Ex: grid-column-start: 1;
    3. Also can do things like grid-column: 1 / span 2;
       1. Starts at position one, spans two wide
    4. And things like: grid-column: 1 / -1;
       1. Starts at position one, spans across the entire width
       2. Doesn’t work with grid-row unless explicitly defined
    5. ‘auto-fill’ says “I don’t know how many columns I want, you figure it out
       1. Ex: grid-template-columns: repeat(auto-fill, 150ps);
    6. ‘auto-fit’ ends the grid when there isn’t enough items
    7. ‘minmax()’ allows you to set minimum/maximum widths to aid with dynamic window sizes
       1. ‘auto-fit’ and ‘auto-fill’ with minmax() works great!
    8. ‘fitcontent()’ is like ‘auto’ with a max value, pass the max val as an argument

1. **100% vs 100vw, and 100% vs 100vh**
   1. **% fills div, vw/vh fills screen**

**CSS “Style” Attributes**

Format: <tagname style=”property:value;”>

|  |  |  |
| --- | --- | --- |
| **Property** | **Values** | **Description** |
| background-color | red, blue, etc. | Background Color |
| background-image | url(“link.com/image.jpg”) | Place an image in the background |
| background-size | cover, contain, or 100px 200px; |  |
| border | 2px solid Tomato; | Value format is border width type color; |
| color | red, blue, etc. | Text Color |
| display | none, inline, block, inline-block | Modifies display behavior |
| font-family | verdana, courier, etc. | Font Style |
| font-size | 160%, 300%, etc. | Font Size |
| font-style | normal, italic, oblique | Font Style |
| font-weight | normal, bold, 900 | Font Weight (boldness) |
| margin | 50px, 25px, 1rem, etc. | Defines the space outside a border |
| padding | 30px, 20px, 1rem, etc. | Defines the space between text & border |
| text-align | center, left?, etc. | Horizontal Text Alignment |