Basic CSS

Created By	Ⓝ Nate L
Last Edited	@Dec 27, 2018 1:11 PM

CSS can be applied in three ways: inline, within style tags on an HTML doc, or using an external style sheet.

- Inline Method
    - Add a style attribute to an element
    - Declare CSS properties and their values
    - Use a semicolon at the end of each property

    ```
    <h2 style="color: red; font-style: italic;">CatPhotoApp</h2>
    ```

- Style Block Method
    - CSS selectors describe where to apply styling
        - Type selectors (elements)
        - Class selectors
            - The class attribute can be added to any element
            - More specific than elements
        - Attribute selectors
            - Apply styling to elements that have a certain attribute and value
        - Id selectors
            - More specific than classes, because an id is unique to a single element
    - The style block typically goes in the head of the HTML document
    - Use a semicolon at the end of each style property

```
<style>
  h2 {
    color: blue;
  }
  .red-text {
    color: red;
  }
  #id-name {
    color: gold;
  }
  [type="checkbox"] {
    color: green;
  }
</style>
```

- Fonts
  - You can import a font link before the styling element to use fonts that aren't built-in
    - Wrap font names in quotes if they have spaces (and are imported)
  - You can specify a backup font in case the primary font is broken

```
p {
    font-size: 16px;
    font-family: serif, monospace;
}
```

- Useful styling properties are padding, margin, and border
  - These values can also be negative, reducing space between elements
  - There are several shorthand versions for these

```
/* Applies to all sides */
margin: 22px;
padding: 12px;

/* vertical | horizontal */
margin: 5px 8px;
padding: 14px 10px;

/* top | horizontal | bottom */
margin: 15px 10px 4px;
padding: 5px 0 14px;
```

```
/* top | right | bottom | left */
margin: 16px auto 0 30px;
padding: auto 0 10px 6px;
```

- However, it might be easier to just select the specific side of the margin/padding of an element
- As for border, the shorthand version refers to the width, style, and color

```
p {
    border-width: 2px;
    border-style: solid;
    border-color: green;
}

/* can be shortened like so: */

p {
    border: 2px solid green;
}
```

- Pixels (**px**) are an absolute value when it comes to font size, there are other values ***rem and em,*** which are relative units.
    - In terms of font-size, em will inherit the value that is stored in the selected element's parent
    - rem on the other hand, will inherit the size that is defined in the root element, the html element
- Fonts can be specified by using the font-family property
    - If declared in the parent element, the children of the element will inherit the same font (hence the name cascading style sheets, this also applies to other properties)
    - Note that inherited values may be overwritten in child elements
- When creating classes for elements, do note that you can have multiple classes per element, ex: `<p class="first-class second-class">Text</p>`
    - A space between classes signifies a new class, using - or _ will prevent creating a new class

- CSS reads rules from top to bottom, if there is an existing rule that conflicts with another, CSS will overwrite it with the most recently read rule
    - Another quality that overrides rules is specificity (would recommend reading up on that)
        - https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity (MDN is a great resource)
- To style an element with an id selector, use #, ex: class = `.headline`, id: = `#headline`
    - Styling an id will override class styles
    - Furthermore, an inline style will override an id selector
- Another way to override CSS is to use !important, this gives the selector more weight than it naturally has
    - It should be placed at the end of a declaration otherwise it won't have an effect, ex: `font-weight: 3em !important;`
    - Only use it if it's absolutely necessary
    - Use cases would be to aid or test accessibility, temporarily fix an urgent problem, override styles in developer tools, override inline styles in user generated content, etc.
- Hexadecimal values may also be used to represent colors
    - Values break down into RBG: ex: `background-color: #006400;` represents the green in the header for FreeCodeCamp
    - Every two digits represent R, G, or B: #R(00) | G(64) | B(00)
    - These values range from 0-9, A-F which equate to 10 - 15
    - Hex values may also be shortened, ex: Red's value is normally `#FF0000`, which can be shortened to `#F00`, Cyan: `#00FFFF` ⇒ `#0FF`
- In addition to hex values, we can use the RGB property which uses numerical values, ex: to create orange we would use: `background-color: rgb(255, 165, 0);`
- CSS allows for the use of variables, which may be used repeatedly

  - They are available to any elements nested within the element you declare it in
  - Because of this, they are often declared in the `:root` element that way the variables are available throughout the whole web page
- In order to create a CSS variable add two dashes in front of the name you assign to it

```
To create:

:root {
  --main-font-size: 2em;
  --main-color: blue;
}

To use it:

.another-class {
  font-size: var(--main-font-size);
  color: var(--main-color);
}
```

- You may override a variable that is set in the `:root` element by setting it again in a specific element

```
:root {
/* variable declaration */
  --large-text: 2em;
}

.class {
/* override variable */
  --large-text: 3em;
}
```

- Styles won't be applied unless the variable names are an exact match
- You can attach fallback values to variables
  - ex: `color: var(--headline-color, black)`
- In order to improve browser compatibility, provide fallbacks just in case it can't load the initial value, this can be done by having a more widely

supported declaration before your chosen declaration, ex:

```
/* variable declared here */
.text {
  color: yellow:
  color: var(--highlighted);
}
```

- Media queries can be used to set break points in order to change CSS to fit that particular screen size; you can use multiple media queries in order to create multiple break points ex:

```
/* For mobile first design, set a max value for the smallest breakpoint */
@media (max-width: 350px) {
  /* declarations here are set for 350px wide and below */
}

@media (min-width: 351px) and (max-width: 750px) {
  /* declarations here are set for 351px to 750px */
}

/* Use a min value to specify the last breakpoint */
@media (min-width: 751px) {
  /* declarations here are set for 751px and above */
}
```

- You can create more advanced media queries, check out the MDN resource