CSS Notes

As described in the HTML notes, CSS can be written **inline** with the HTML code, but it can also be written in its own CSS document with the **CSS ruleset**. In the files and folders of your computer, you can insert a file with a name and ending in “**.css”**.

**Ruleset terms** for within a CSS document:

* **Selector** – The beginning of the rule set used to target the element that will be styled. This is the first thing written down when trying to set add things to a certain area of the HTML attached. Examples could include **p, h1, #id, or .class**.
* **Declaration Block** – The code in-between (and including) the curly braces ({ }) that contains the CSS declaration(s).
* **Declaration** – The group name for a property and value pair that applies a style to the selected element. This is basically what’s written in between the curly braces which includes two parts: the **property** and the **value**.
* **Property** – The first part of the declaration that signifies what visual characteristic of the element is to be modified.
* **Value** – The second part of the declaration that signifies the value of the property.

To connect the HTML and CSS files, you need to use a **<link>** element. This element must be placed within the head of the HTML file. It is a self-closing tag. Within the <link> element, two attributes are needed: **href= “url”**  and **rel= “…”**.

The value of the **href** must be an address, or path, to the CSS file. Using terminal/command prompt language, you type in the location of the CSS file. If the CSS file is on the same level as the HTML file, then just type in the file name of the CSS.

**rel** is the attribute that describes the relationship between the HTML file and the CSS file. For now, it’ll be linked to a **“stylesheet”**.

A **selector** is what decides what a style is being applied to. A “**type” selector** matches the type of element in the HTML. A type is just a specific element name. An example would be **p** within the <p> element.

In order apply a style to every element in the HTML at once, you would use the **universal selector**. The universal selector is just the “\*” symbol in the same place you put the type selector.

The **class** attribute is one of the most common ways to select an element. Instead of selecting every single <p> element, you can give one or multiple <p> elements a class to only style those specific <p> elements. When writing the class selector, you write a period before the value of the class. **Example:**

.example { }

An element can be given **multiple classes** and leads to an interconnected “web” of classes across all the elements. When adding multiple classes, a space will be put in-between each one. **Example:**

<p class=”example1” class=”example2”>

**ID** is another attribute you can use to style specific elements. To select an element’s ID with CSS, we prepend the id name with a number sign (#). Example:

#example { }

Usually, IDs don’t apply to more than one element and are specific to one area.

**IDs and classes** can be mixed since some elements would want the same class as one group of the document, but also might need its own separate style that’s put within its ID.

To target specific **attributes** within an html without adding new code for each attribute needing change, you can just target that specific attribute by surrounding the attribute in brackets ([ ]). If you would like to single out an attribute value, you can use **type[attribute\*=value]**. This code selects an element where the attribute contains any instance of the specified value. Example:

<img src=’/images/winter.jpg’>

<img src=’/images/summer.jpg’>

**CSS that would go along with it:**

img[src\*=’winter’] {

}

Img[src\*=’summer’] {

}

The first CSS example would change any style added (within the curly braces) for the first <img> element only. Same goes with the second CSS example with the second <img> element.

**Pseudo-classes** can be attached after any selector and factors in more situations like user interaction, site navigation, and position in the document tree. It can be written as a colon followed by a name. Some **examples** of these classes would be:

:focus, :visited, :hover, :disabled, :active, and more.

**Specificity** is the order in which CSS styles get applied, ID being the strongest and universal being the weakest. **Order:**

Universal < Tag < Class < ID

**Chaining** might be needed when an HTML element requires two or more CSS selectors at the same time. Example:

h1.special {

}

The code above would only select the **<h1>** elements with a class of **special**.

CSS also supports selecting elements that are nested within other HTML elements, known as **Descendants**. A descendant selected by adding the nested element after the first selector. Example:

<ul class=’main-list’>

<li> … </li>

<li> … </li>

<li> … </li>

</ul>

.main-list, li {

}

In the above example, **.main-list** selects the element with the .main-list class. The descendant <li>’s are selected by adding li to the selector, separated by a space. This resulting code will change the style, specifically, of the list items within the .main list.

If you want to give the same style to multiple elements, you can combine **multiple selectors** to change many styles with just one. **Example:**

h1,

.menu {

font-family: Georgia;

}

All that is needed is a comma and a line to separate the multiple selectors.

**font-family** changes the typeface of the element.

**font-size** controls the size of the text displayed.

**font-weight** defines how thin or thick text is displayed.

**text-align** places text in the left, right, or center of its parent container.

Text can have two different color attributes: **color** and **background-color**. Color defines the color of the text while background-color defines the color behind the text.

CSS can make an element transparent with the **opacity** property.

CSS can also set the background of an element to an image with the **background-image** property.

The **!important** flag will override any style, however it should almost never be used, as it is extremely difficult to override.

The **box model** comprises a set of properties used to create space around and between HTML elements.

The **height** and **width** of a content area can be set in pixels or percentages of the screen.

**Borders** surround the content area and padding of an element. The color, style, and thickness of a border can be set with CSS properties.

**Padding** is the space between the content area and the border. It and be set in pixels or percentages.

**Margin** is the amount of spacing outside of an element’s border.

**Horizontal margins** are added, so the total space between the borders of adjacent elements is equal to the sum of the right margin of one element and left margin of the adjacent element.

**Vertical margins** collapse, so the space between vertically adjacent elements is equal to the larger margin.

**margin: 0 auto** horizontally centers an element inside of its parent content area, if it has a width.

The **overflow** property can be set to **display**, **hidden**, or **scroll**, and dictates how HTML will render content that overflows its parent content area.

The **visibility** property can hide or show elements.

In the default **box model**, box dimensions are affected by border thickness and padding. The **box-sizing** property controls the box model used by the browser. The default value of the box-sizing property is **content-box**. Changing the value to **border-box** makes the width and height contain the margin, border and padding as well as the content’s width and height. This makes it cleaner and easier to calculate the actual width and height of an element’s total box model.

There are 5 basic properties for adjusting the position of HTML elements in the browser:

* **Position**
* **Display**
* **Z-index**
* **Float**
* **Clear**

**Position** is set to static by default. The block-level elements take up their own line of space and therefore don’t overlap with each other. They also consistently appear on the left side of the browser.

**Position**: **Relative** allows you to position an element relative to its default static position on the web page. Although the code instructs the browser to expect a relative positioning, it does not specify where the element should be positioned on the page. This is done by adding an offset property that will move the element away from its default static position:

* **Top** – moves the element down from the top.
* **Bottom** – moves the element up from the bottom.
* **Left** – move the element away from the left side (to the right).
* **Right** – moves the element away from the right side (to the left).

**Position: Absolute** will set an elements position to be ignored by all other elements on the page and will basically act like it isn’t present on the page. Can be given coordinates to be like the relative position. It will not follow the user when the page is scrolled.

**Position: Fixed** is like absolute in that it will make the element it is attached to be ignored by all other elements. Its main difference is that it keeps the element in the same place on the page and moves when the user scrolling up and down the page. This position can be given the same offset properties.

**Position**: **Sticky** keeps an element in the document flow as the user scrolls but sticks to a specified position as the page is scrolled further. This is done by using the **sticky** value along with the familiar offset properties, as well as one new one.

**Z-index** properties control how far back or how far forward an element should appear on the web page when elements overlap. Can be thought of as a depth system, with deeper elements appearing behind shallower elements. The z-index accepts integer values. A higher integer value means that the element will be shallower and appear more in front of other elements.

**Display: Inline** provides the ability to make any element an inline element. This includes elements that are not inline by default such as paragraphs, divs, and headings.

**Display: Block** puts whatever is in the selector into its own block whenever it applies.

**Display: Inline-Block** combines features of both inline and block elements. Inline-Block elements can appear next to each other, and we can specify their dimensions using the width and height properties. Images are the best examples.

The **float** property is commonly used for wrapping text around an image. Note, however, that moving elements left or right for layout purposes is better suited for tools like CSS grid and flexbox. It is often set using one of the values:

* **Left** – moves, or floats, elements as far left as possible.
* **Right** – moves elements as far right as possible.

The float property can be used to move multiple elements at once and sometimes the elements bump together. The **clear** property specifies how elements should behave when they bump into each other on the page:

* **Left** – the left side of the element will not touch any other element within the same containing element.
* **Right** – the right side of the element will not touch any other element within the same containing element.
* **Both** – neither side of the element will touch any other element within the same containing element.
* **None** – the element can touch either side.

**Hexadecimal or hex colors** is a number system that has sixteen digits, 0 to 9 followed by A to F. They always begin with a “#” and specify values of red, blue, and green. Six-digit hex values with duplicate values for each RGB value can be shortened to three digits.

**RGB** colors use the “rgb( )” syntax with one value for red, one for blue, and one for green. RGB values range from 0 to 255 and look like this: rgb(7, 210, 50).

**HSL** stands for hue, saturation, and lightness. Hue ranges from 0 to 360 and saturation and lightness are both represented by percentages.

You can add **opacity** to color in RGB and HSL by adding a fourth value, **a,**  which is represented as a percentage.

**Typography** is the art of arranging text on a page.

Text can appear bold or thin with the **font-weight** property.

Text can appear in italics with the **font-style** property.

The vertical spacing between lines of text can be modified with the **line-height** property.

**Serif fonts** have extra details at the end of each letter. **Sans-Serif** fonts do not.

**Fallback fonts** are used when a certain font is not installed on a user’s computer. Sort of like a backup font in case one doesn’t work.

The **word-spacing**  property changes how far apart individual words are while the **letter-spacing** property changes how far apart individual letters are.

**Google Fonts** provides free fonts that can be used in an HTML file with the **<link>** tag or the **@font-face** property.

**Local fonts** can be added to a document with the @font-faceproperty and the path to the font’s source.