***CSS Introduction***

* CSS stands for Cascading Style Sheets
* CSS describes how HTML elements are displayed on screen, paper or in other media.
* CSS saves a lot of work. It can control the layout of multiple web pages all at once.
* External stylesheets are stored in CSS files.

*Why use CSS?*

CSS is used to define styles for our web pages, including the design, layout and variations in display for different devices and screen sizes.

*CSS solved a big problem:*

HTML was never intended to contain tags for formatting a web page HTML was created to describe the content of a web page, like:

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

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

When tags like <font> and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process. To solve this problem, the World Wide Web Consortium (W3C) created CSS.

CSS removed the style formatting from the HTML page.

*CSS Saves a lot of Work:* The style definitions are normally saved in external .css files. With an external stylesheet file, you can change the look and feel of an entire website by changing just one file.

How to add CSS?

There are three ways to add CSS to our HTML page:

1. Inline CSS <tag style=”css” />
2. Internal CSS <style>css</style>
3. External Stylesheet <link href=”style.css”/>

Inline CSS is useful, when we need to add styling only for a single element on the web page.

Internal CSS is useful when we are adding styles to a single page website or only for one web page.

External CSS is the most crucial and is used frequently, as it supports styling for multi-page websites. We can use a single CSS file that can be embedded into multiple HTML documents to provide styling.

Examples of all three ways to add CSS is:

index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>CSS</title>

    <link rel="stylesheet" href="style.css">

    <style>

        p{

            background-color: blueviolet;  /\* internal css \*/

        }

    </style>

</head>

<body>

    <h1 style="background-color: burlywood;"> <!-- inline CSS -->

        hello inline css

    </h1>

    <p>

        hello internal css.

    </p>

    <div>

        hello external css

    </div>

</body>

</html>

style.css

div{

    background-color: brown;

}

Output:



***CSS Selectors***

CSS Selectors are used to “find” or select the HTML elements you want to style. There are 5 types of selectors:

1. Element/Type selector
2. Class Selector
3. Id selector
4. Attribute Selector
5. Universal selector

Element/Type Selector: The CSS type/element selector matches elements by node name. in other words, it selects all elements of the given type within a document.

Class Selector: The CSS class selector matches elements based on the contents of their class attribute.

Id Selector: The CSS id selector matches elements based on the contents of their id attribute.

Attribute Selector: The CSS attribute selector matches elements based on the element having a given attribute explicitly set, with options for defining an attribute value or substring value match.

Universal Selector: The CSS universal selector matches elements of any type.

We need to achieve the following:

A close-up of a text

Description automatically generated

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>CSS</title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <!-- selectors example -->

    <h1>CSS Selectors</h1>

    <h2>Applying CSS to Different Parts of HTML</h2>

    <!-- TODO 1: Set the CSS for all paragraph tags to "color: red" -->

    <p class="note">1. The element selector targets elements based on their HTML tag name.</p>

    <ol>

        <!-- TODO 2: Set the CSS for all elements with a class of "note" to "font-size: 20px" -->

        <li class="note" value="2">Class selectors target elements based on the value of the class attribute.</li>

        <!-- TODO 3: Set the CSS for the element with an id of "id-selector-demo" to "color: green" -->

        <li class="note" id="id-selector-demo" value="3">ID selectors target elements based on the value of the id

        attribute.</li>

        <!-- TODO 4: Set the CSS for the li elements that have the "value" attribute set to "4" to have "color: blue" -->

        <li class="note" value="4">Attribute selectors target elements based on their attributes and values.</li>

        <!-- TODO 5: Set all elements to have "text-align: center" -->

        <li class="note" value="5">The universal selector targets all elements.</li>

    </ol>

</body>

</html>

Style.css

ol {

    margin-left: -40px;

    margin-top: -20px;

    list-style-position: inside;

  }

/\* TODO 1: Set the CSS for all paragraph tags to "color: red" \*/

p{

    color: red;

}

/\* TODO 2: Set the CSS for all elements with a class of "note" to "font-size: 20px" \*/

.note{

    font-size: 20px;

}

/\* TODO 3: Set the CSS for the element with an id of "id-selector-demo" to "color: green" \*/

#id-selector-demo{

    color: green;

}

/\* TODO 4: Set the CSS for the li elements that have the "value" attribute set to "4" to have "color: blue" \*/

li[value="4"]{

    color: blue;

}

/\* TODO 5: Set all elements to have "text-align: center" \*/

\*{

    text-align: center;

}

Project: Color Vocab Website

* Code is in the repository.

***CSS Properties***

**Color Properties:** we use two types of color properties, the “background-color” and “color”. The background color property is used to specify the shade/color we want for background, and the color property is used to set the color of the text.

Example: Achieve the following using color properties:

A close up of a line

Description automatically generated with medium confidence

Code is in CSS folder in outside index.html and style.css.

**Font Properties:** there are different font properties available in CSS, like “font-size”, “font-weight”, “font-family”, etc. “color” is also a type of font property, as it changes the text/font color.

1. “font-size”: there are different ways in which we can define the font size, such as px(pixels), pt(points), em() and rem().

* 1px: 1 pixel is 1/96th of an inch. This means around 0.26 mm. This is the size that it takes up as a square pixel.
* 1pt: 1 point is the 1/72th of an inch, this means around 0.35 mm, little bit larger than the pixel. While writing the word document, we use the points, like if font size is selected to be 12, then it represents 12 points.
* 1em: em is pronounced as “m” letter. This em is 100% of its parent, for example, if we are applying this property as “h1: {font-size: 1 em; }” and this h1 is enclosed into the body tag, then it will take up the value related to the body tag, say body tag is having 20px size, so 1 em will be equal to “100% of parent” i.e., 20px, while if we set h1 to 2em, then it will be 2\*100% of parent, i.e., 2\*20px = 40px.
* 1rem: rem is also a relative property, but it is “100% of root” and we know that “html” element is our root element. So, rem takes up the size relative to html element.

Example of em vs. rem: let’s say we have below code snippet:

<html>

    <body>

        <h1>hello</h1>

        <footer>

            <h2>there</h2>

        </footer>

    </body>

</html>

1. I am setting the font-size of footer as 20px and font-size of h2 as 2em. So, h2 will take the size relative to its parent in this case, i.e., of footer element. So, 2em = 2\*20px = 40px.
2. Now in this case, font-size of footer is same as above, 20px, and font-size of h2 this time is 2rem. Now, this 2rem will not be affected, if I change the font-size of footer, because it takes up the size relative to its root element, and here root is html element, so if font-size of html is defined to be 10px let’s say, then 2rem will be 2\*10px = 20px.

NOTE: it is recommended to use rem while defining the font-size of any element, because root is only one for each case, while parent can be multiple elements. In the above example, we can see, h2 has footer as its parent element, footer has body and body has html as its parent element. So, this might be a bit confusing, and can lead to inconsistency in the webpages.

1. “font-weight”: we can specify font-weight into three ways – using keywords (bold, normal, etc.), using numbers (100 to 900) and relative to parent (lighter (-100), bolder (+100), etc.)
2. “font-family”: font-family is used to specify the typeface or the font style we want in our website. These can be of two types, “serif” and “sans-serif”. Below is the difference between them.

|  |  |
| --- | --- |
| A black text on a white background  Description automatically generated | A black text on a white background  Description automatically generated |

To include custom fonts in our website, we can go to “fonts.google.com” and can choose any font of our choice and can embed that in our website.

1. “text-align”: This property allows us to manage the text position, where we want our text to be, it has different values available, like – center, start, end, left, right, justify, etc.

Example:

Index.html

<!-- 6.1 font properties -->

    <!-- to-do:

        1. change the color of <p>color</p> to coral color

        2. change the font size of <p>font size</p> to 2X the size of the root font size.

        3. change the font weight of <p>font weight</p> to 900

        4. change the font family of <p>font family</p> to the Google font Caveat with regular(400) font weignt.

        5. change the <p>text align</p> to right aling.

        6. change the root (html element) font size to 30px.

    -->

    <h1>Important CSS properties</h1>

    <p id="color">color</p>

    <p id="size">font size</p>

    <p id="weight">font weight</p>

    <p id="family">font family</p>

    <p id="alignment">text align</p>

Style.css

#color{

      color: coral;

  }

  #size{

      font-size: 2rem;

  }

  #weight{

      font-weight: 900;

  }

  #family{

    font-family: 'Caveat', cursive;

  }

  #alignment{

      text-align: right;

  }

  html{

      font-size: 30px;

  }

Output:



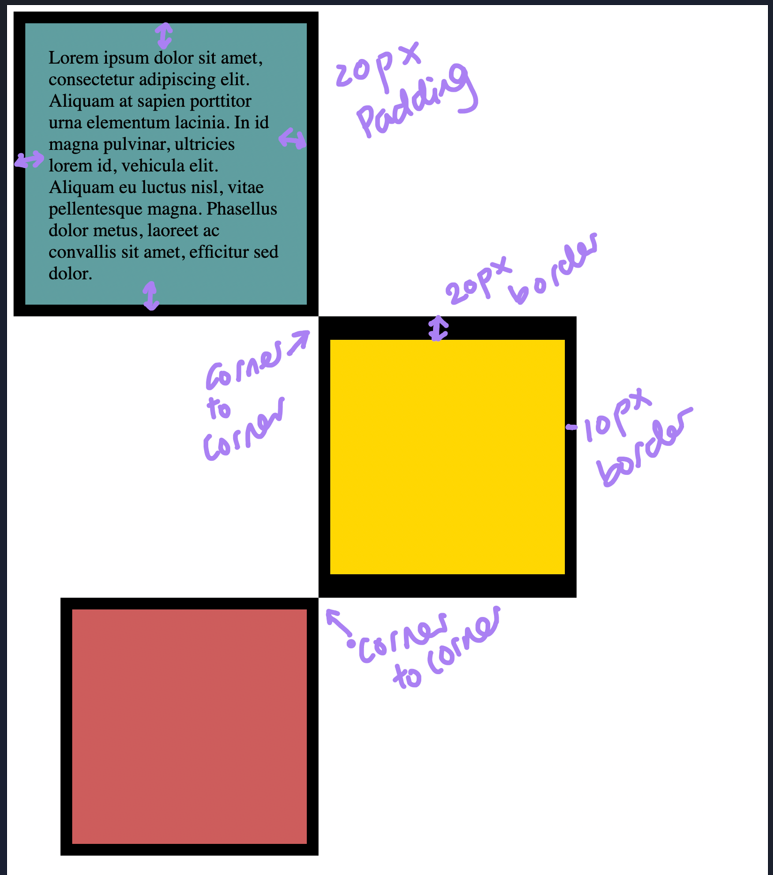
**Inspecting CSS:** we can try and play around using Chrome developer tools.

**The CSS Box Model – Margin, Padding and Border:** The CSS box model is essentially a box that wraps around every HTML element. It consists of margin, border, padding and the actual content. It defines how all these different parts work.

A diagram of a website

Description automatically generated with medium confidence

Complete the below task to get some hands-on experience on the box model. We need to achieve the following:



Code:

index.html

<!-- 6.3 box model in CSS -->

    <!-- to-do:

        1. create 3 boxes using the div element

        2. set their sizes to 200px high by 200px wide.

        3. set different background colors for each of the boxes

        4. add a paragraph element to first div element and add some content to it

        5. set the first div to have 20px padding all around with a black 10px border

        6. fix the style of <p> element to remove all margins.

        7. set the second div to have a 20px border on top and bottom and 10px border on left and right.

        8. set the third div to have a 10px border.

        9. set the margins for the dives so that each box corner touches the other.

    -->

    <div id="first">

        <p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Consectetur blanditiis quidem magni explicabo excepturi commodi harum repellendus laudantium! Iure excepturi, eos non incidunt assumenda officiis sit beatae suscipit minima voluptates.</p>

    </div>

    <div id="second"></div>

    <div id="third"></div>

style.css

  div{

      width: 200px;

      height: 200px;

  }

  p{

      margin: 0;

  }

  #first{

    background-color: aquamarine;

    padding: 20px;

    border: 10px solid black;

  }

  #second{

    background-color: bisque;

    border: solid black;

    border-width: 20px 10px;

    margin-left: 260px;

  }

  #third{

    background-color: burlywood;

    border: 10px solid black;

    margin-left: 40px;

  }

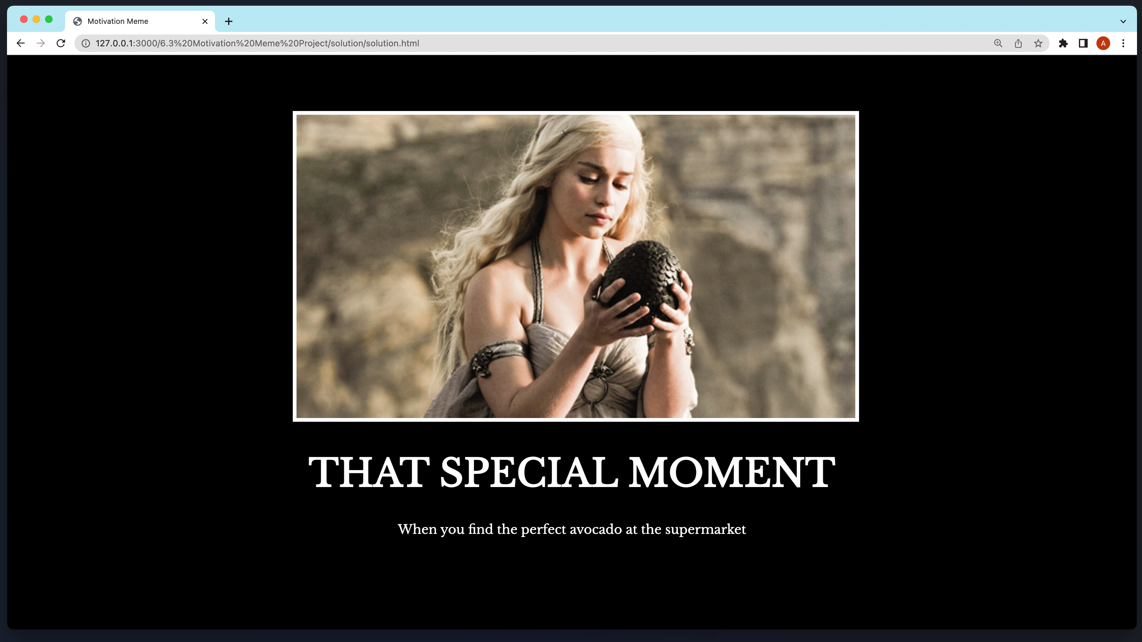
Output:

A close-up of a screen

Description automatically generated

***Project: Motivational Poster Website***

In this project we are required to achieve the following goal:



Code:

index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Motivational Poster Image</title>

    <link rel="stylesheet" href="./style.css">

    <link rel="preconnect" href="https://fonts.googleapis.com">

    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

    <link href="https://fonts.googleapis.com/css2?family=Libre+Baskerville&display=swap" rel="stylesheet">

</head>

<body>

<!--

  TODO: Create a motivational post website.

    Style it how ever you like.

    Look at the goal image for inspiration.

    But it must have the following features:

    1. The main h1 text should be using the Regular Libre Baskerville Font from Google Fonts:

    https://fonts.google.com/specimen/Libre+Baskerville

    2. The text should be white and background black.

    3. Add your own image into the images folder inside assets. It should have a 5px white border.

    4. The text should be center aligned.

    5. Create a div to contain the h1, p and img elements. Adjust the margins so that the image and text are centered on the page.

    Hint: You horizontally center a div by giving it a width of 50% and a margin-left of 25%.

    Hint: Set the image to have a width of 100% so it fills the div.

    6. Read about the text-transform property on MDN docs to make the h1 uppercase with CSS.

    https://developer.mozilla.org/en-US/docs/Web/CSS/text-transform

-->

<div class="poster">

  <img src="./assets/images/daenerys.jpeg" alt="girl image" class="image">

  <h1>that special moment</h1>

  <p>When you find the perfect avocado at the supermarket</p>

</div>

</body>

</html>

style.css

body{

    background-color: black;

    color: white;

}

h1{

    text-transform: uppercase;

    font-size: 3rem;

}

.image{

    border: 5px solid white;

    width: 100%;

}

.poster{

    font-family: 'Libre Baskerville', serif;

    width: 50%;

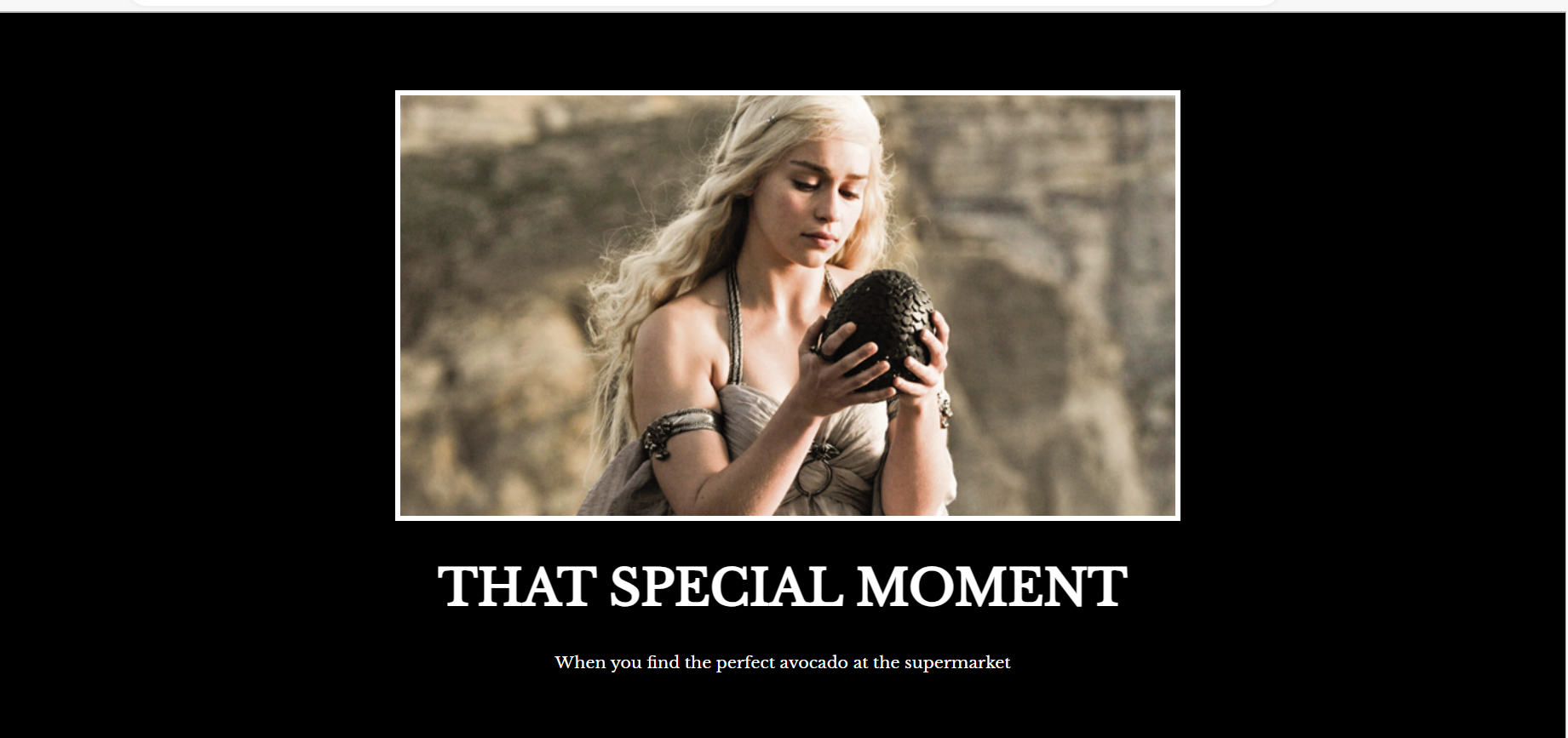
    margin-left: 25%;

    text-align: center;

    margin-top: 5%;

}

Output:



***Intermediate CSS***

***The Cascade***: sometimes two or more competing CSS rules could apply to an element. The cascade is the algorithm for solving conflicts where multiple CSS rules apply to an HTML element.

Example:

index.html

<h1>hello there</h1>

style.css

h1{

    color: red;

}

h1{

    color: green;

}

Above CSS is conflicting, that what color will h1 element is going to take? To resolve this conflict, we have the concept of Cascade. It says, the CSS rule which is the most recent in above type of case, will be applied to it. So we can understand that in the above case, h1 will take green color.

Output:

A green and white text

Description automatically generated

Understanding the cascade algorithm helps us understand how the browser resolves conflicts like this. The cascade algorithm is split into 4 distinct stages.

1. Position and order of appearance: the order of which our CSS rules appear (the above example)
2. Specificity: an algorithm which determines which CSS selector has the strongest match
3. Origin: the order of when CSS appears and where it comes from, whether that is a browser style, CSS from a browser extension, or authored CSS
4. Importance: some CSS rules are weighted more heavily than others, especially with the “!important” rule type

**Position and order of appearance:** The order in which our CSS rules appear and how they appear is taken into consideration by the cascade while it calculates conflict resolution.

Styles can come from various sources on an HTML page, such as <link> tag, an embedded <style> tag, and inline CSS as defined in an element’s style attribute.

1. If we have a <link> that includes CSS at the top of HTML page, then another <link> that includes CSS at the bottom of the page: the bottom <link> will have the most specificity. The same thing happens with embedded <style> elements, too. They get more specific, the further down the page they are.
2. The ordering also applies to embedded <style> elements. If they are declared before a <link>, the linked stylesheet’s CSS will have the most specificity.

Example: The <style> element is declared in the <head> while the <link> element is declared in the <body>. This means that the <link> element gets more specificity than the <style> element.

1. An inline <style> attribute with CSS declared in it will override all other CSS, regardless of its position, unless a declaration has !important defined.
2. Position also applies in the order of our CSS rule. In below example, the element will have a purple background because background: purple was declared last. Because the green background was declared before the purple background, it is now ignored by the browser.

/\* cascade: postion and order example \*/

.my-element{

    background-color: green;

    background-color: purple;

}

1. Being able to specify two values for the same property can be a simple way to create fallbacks for browsers that do not support a particular value. In the example below, font-size is declared twice. If clamp() is supported in the browser, then the previous font-size declaration will be discarded. If clamp() is not supported by the browser, the initial declaration will be honored, and the font-size will be 1.5rem.

.my-element {

    font-size: 1.5rem;

    font-size: clamp(1.5rem, 1rem + 3vw, 2rem);

}

**Specificity**: Specificity is an algorithm which determines which CSS selector is the most specific, using a weighting or scoring system to make those calculations. By making a rule more specific, you can cause it to be applied even if some other CSS that matches the selector appears later in the CSS.

CSS targeting a class on an element will make that rule more specific, and therefore seen as more important to be applied, than CSS targeting the element alone. This means that with the following CSS, the h1 will be colored red even though both rules match the rule for the h1 selector comes later in the stylesheet.

HTML:

<h1 class="my-element">Heading</h1>

CSS:

.my-element {

    color: red;

}

h1 {

    color: blue;

}

Output:

A red text on a white background

Description automatically generated

**Origin**: the CSS that we write isn’t the only CSS applied to a page. The cascade takes into account the origin of the CSS. This origin includes the browser’s internal stylesheet, styles added by browser extensions or the operating system, and your authored CSS (CSS defined by the web developer). The order of specificity of these origins from least specific to most specific is as follows:

1. User agent base styles: these are the styles that your browser applies to HTML elements by default.
2. Local user styles: These can come from operating system level, such as a base font size, or a preference of reduced motion. They can also come from browser extensions, such as a browser extension that allows a user to write their own custom CSS for a webpage.
3. Authored CSS: the CSS that you author.
4. Authored !important: any !important that you add to your authored declarations.
5. Local user styles !important: any !important that come from the operating system level, or browser extension level CSS.
6. User agent !important: any !important that are defined in default CSS, provided by the browser.

A close-up of a computer code

Description automatically generated

If we have an !important rule type in the CSS we have authored and the user has an !important rule type in their custom CSS, then user style will win.

A screenshot of a computer

Description automatically generated

**Importance:** not all CSS rules are calculated the same as each other or given the same specificity as each other. The order of importance, from least to most important is as follows:

1. Normal rule type, such as font-size, background or color
2. Animation rule type
3. !important rule type (following the same order as origin)
4. Transition rule type

Active animation and transition rule types have higher importance than normal rules. In the case of transitions higher importance than !important rule types. This is because when an animation or transition becomes active, its expected behavior is to change visual state.

A screenshot of a computer

Description automatically generated

Specificity: suppose that we are working with the following HTML and CSS:

<button class="branding">Hello, Specificity!</button>

button {

    color: red;

  }

  .branding {

    color: blue;

  }

Output:

A close-up of a sign

Description automatically generated

There are two competing rules here. One will color the button red and the other will color it blue. Which rule gets applied to the element? Understanding the CSS specification’s algorithm about specificity is the key to understanding how CSS decides between competing rules.

Specificity scoring: each selector rule gets a score. We can think of specificity as a total score and each selector type earns points towards that score. The selector with the highest score wins.

Scoring each selector type: each selector type earns points. We add all of these points up to calculate a selector’s overall specificity.

1. Universal selector: A universal selector (\*) has no specificity and gets 0 points. This means any rule with 1 or more points will override it. For example, we have specified text color of text on website to be black (using universal selector, so that it applies to all) but we wanted that h1 element should have text color as cyan, so when we will specify the h1 element having color to be cyan, then cyan will be applied, as it will earn a point, and will override the universal selector’s style.
2. Element or pseudo-element selector: an element (type) or pseudo-element selector gets 1 point of specificity.
3. Type selector

button {

    color: red;

  }

1. Pseudo-element selector

::selection{

    color: green;

}

1. Class, pseudo-class or attribute selector: a class, pseudo-class or attribute selector gets 10 points of specificity.
2. Class selector

.my-class {

    color: red;

}

1. Pseudo-class selector

:hover {

    color: red;

}

1. Attribute selector

[href='#'] {

    color: red;

}

1. The :not() pseudo-class itself adds nothing to the specificity calculation. However, the selectors passed in as arguments do get added to the specificity calculation.

div:not(.my-class) {

    color: red;

}

The above sample would have 11 points of specificity because it has one type selector div and one class inside the :not().

**Id selector:** an id selector gets 100 points of specificity, as long as we use an id selector #id\_name and not attribute selector [id=”id\_name”].

#id-name{

    background-color: aliceblue;

}

**Inline style attribute:** CSS applied directly to style attribute of the HTML element, gets a specificity score of 1000 points. This means that in order to override it in CSS, you have to write an extremely specific selector.

<div style="color: red"></div>

**!important rule:** Lastly, an !important rule at the end of a CSS value gets a specificity score of 10000 points. This is the highest specificity that one individual item can get. An !important rule is applied to a CSS property, so everything in the overall rule (selector and properties) does not get the same specificity score.

.my-class {

    color: red !important; /\* 10,000 points \*/

    background: white; /\* 10 points \*/

}

A screenshot of a computer

Description automatically generated

***Specificity in context:*** the specificity of each selector that matches an element is added together. Considering below example:

<a class="my-class another-class" href="#">A link</a>

What happens if we provide different CSS rules:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| This link has two classes on it. Add the following CSS and it gets 1 point of specificity: | Reference one of the classes in this rule, it now has 11 points of specificity: | Add the other class to the selector, it now has 21 points of specificity: | Add the href attribute to the selector, it has now 31 points of specificity: | Finally, add a :hover pseudo-class to all of that, the selector ends up with 41 points of specificity: |
| a {  color: red; } | a.my-class {  color: green; } | a.my-class.another-class {  color: rebeccapurple; } | a.my-class.another-class[href] {  color: goldenrod; } | a.my-class.another-class[href]:hover {  color: lightgrey; } |

Visualizing specificity: in diagrams and specificity calculators, the specificity is often visualized like this:

A screenshot of a computer

Description automatically generated

The left group is id selectors. The second group is class, attribute and pseudo-class selectors. The final group is element and pseudo-element selectors. For reference, the following selector is 0-4-1:

a.my-class.another-class[href]:hover {

    color: lightgrey;

}

A screenshot of a computer

Description automatically generated

Pragmatically increasing specificity: Let’s say we have some CSS that looks like this:

.my-button {

    background: blue;

}

  button[onclick] {

    background: grey;

}

With HTML that looks like this:

 <button class="my-button" onclick="alert('hello')">Click me</button>

The second CSS rule will win, as it will get 1 more point. So the button will have a grey background, because the second selector earns 11 points of specificity (0-1-1). This is because it has one type selector button, which is 1 point and an attribute selector [onclick], which is 10 points.

The previous rule - .my-button – gets 10 points (0-1-0), because it has one class selector.

If we want to give the first rule a boost, repeat the class selector like below:

.my-button.my-button {

    background: blue;

}

button[onclick] {

    background: grey;

}

# Repetition boosts specificity, and blue background will be applied to button background. Because, the new selector gets a score of 20 points (0-2-0).

***A matching specificity score sees the newest instance win***: let’s stay with the button example for now and switch CSS around to this:

.my-button {

    background: blue;

  }

[onclick] {

    background: grey;

  }

The button will have a grey background, because both selectors have an identical specificity score (0-1-0).

***Inheritance:*** some CSS properties inherit if we don’t specify a value for them. Say we just wrote some CSS to make elements look like a button.

<a href="http://example.com" class="my-button">I am a button link</a>

.my-button {

    display: inline-block;

    padding: 1rem 2rem;

    text-decoration: none;

    background: pink;

    font: inherit;

    text-align: center;

  }

We then add a link element to an article of content, with a class value of .my-button. However there’s an issue, the text is not the color that you expected it to be.