**CSS (Cascading style sheet):**

**1.Inline CSS:**

value

**<html style=”background: blue”>**

property

**</html>**

**Inline CSS will go with the opening tag of the element.**

**This code will give the blue colour background in Inline CSS.**

**Inline elements are really useful for adding CSS style to just a single element on your HTML page.**

**It’s not normally recommended to use inline styles in your entire document. It’s only for specific sections, or when you’re testing, or when you only want it in one single element or one line in your HTML document.**

**2. Internal CSS:**

**Internal CSS is done through a special HTML tag called the “Style element.”**

**<html>**

**<head>**

**<style>**

**html {**

**Background: red;**

**}**

**</style>**

**</head>**

**</html>**

**We have got the open<style> tag and closing </style> tag.**

**In this between those two lines is where we add all our CSS.**

**Internal styles are really useful for applying it only to one HTML.**

**HOWEVER, if we have a multi-page website, then you probably shouldn’t be using the internal style, instead we should be using something called an “External CSS styling.”**

**Index.html**

**<html>**

**<head>**

**<link**

relationship

**rel=”stylesheet”**

**href=”./styles.css”**

location

**/>**

**</link>**

**</head>**

**</html>**

**Styles.csss**

**html {**

**background:green;**

**}**

**This is the most common way to use the CSS.**

**Summary:**

**Inline: use in single element.**

**Internal: use in single webpage**

**External: use in website or multi webpages.**

**CSS Selectors:**

**we can create CSS rules by simply specifying two things.**

**One is the property we want to change and after a colon we get to specify the value to change that property.**

**E.g.;**

**Colour: blue; here the colour is the property and blue is the value**

**h1 or any tag element will be the selector to bring style or CSS here in h1 will have the effect of having blue colour in text**

**CSS Selector**

**h1 {**

**background: green;**

**}**

**What is CSS Selector? -> well, it’s the part that selects the HTML in order to apply whichever rules go in between these curly**

**Class Selector**

**.red-heading {**

**Color: red**

**}**

**Class is something that we can add as an attribute to any html element. Class will allow to group all the elements which will show the effect. here only first h2 element will has its effect but not others h2.**

**Index.html**

**<h2 class =”red-text”>Red</h2>**

**<h2>Green</h2>**

**<h2>Blue</h2>**

**Styles.css**

**.red-text {**

**Color: red;**

**}**

**Also, we can pass the same name to other tag and elements as well it won’t have the problem.**

**<h2 class =”red-text”>Heading 2</h2>**

**<h3>Heading 3</h3>**

**<p class=”red-text”>Paragraph</p>**

**Here both the h2 and p will have the red colour**

**You can create a selector that will target specific elements with the class applied.**

**h2. red-text {**

**colour: red;**

**}**

**Heading (h2) will red in colour**

**p. red-text {**

**colour: yellow;**

**}**

**Now the paragraph(p) will have yellow colour. So even though the class is same for all selector like h1 and p, using dot-notation we can apply colour to the specific class.**

**ID Selector:**

**Id selector, it has its own special symbol, which is a pound or hastag (#)**

**No spaces in between the pound sign and the actual name of the ID, and this selects all elements.**

**Index.html**

**<h2 id=”main”>Red</h2>**

**<h2>Green</h2>**

**<h2>Blue</h2>**

**Styles.css**

**#main {**

**Color: red;**

**}**

**Similar to the class selector id selector will work same like id with specific element with name and in CSS also has the same name after # and it works.**

**So, then what is the difference between the ID and the class selector?**

**Well, the class selector can be applied to many elements, whereas the ID should be only applied to one element in a single HTML file. In a single HTML file like our index.html, there should only be one ID of this particular name, main and it should be completely unique and this is the difference.**

**NOTE; IDs are unique only one element per file and classes, you can put on as many elements as you like to group them together.**

**Key Differences:**

1. **Uniqueness**:
   * **ID**: Must be **unique** on a page (one instance per ID).
   * **Class**: Can be reused **multiple times** across elements.
2. **Specificity**:
   * **ID** has **higher specificity** than classes. Styles applied via an ID will override conflicting class styles.
3. **Syntax**:
   * **ID**: Defined with # in CSS (e.g., #header).  
     HTML: <div id="header"></div>.
   * **Class**: Defined with . in CSS (e.g., .button).  
     HTML: <button class="button"></button>.
4. **Use Cases**:
   * **ID**: Best for **unique elements** (e.g., a header, footer, or specific section).
   * **Class**: Best for **reusable styles** (e.g., buttons, cards, error messages).

**When to Use ID:**

1. **Unique Elements**: For one-off components (e.g., #contact-form, #main-nav).
2. **Fragment Links**: To link to specific page sections (e.g., href="#section-2").
3. **JavaScript Targeting**: When using getElementById() for DOM manipulation.
4. **Accessibility**: For ARIA attributes like aria-labelledby or aria-describedby.

**When to Use Class:**

1. **Reusable Styles**: For styling multiple elements (e.g., .card, .alert).
2. **Component-Based Design**: When creating shared UI components (e.g., grids, buttons).
3. **Lower Specificity**: To avoid specificity conflicts and keep CSS maintainable.
4. **State Styling**: For dynamic states (e.g., .active, .hidden).

**Best Practices:**

* **Avoid IDs for Styling**: Prefer classes for CSS to minimize specificity issues. Use IDs for JavaScript hooks or fragment links.
* **Keep Specificity Low**: Use classes and avoid nesting to make overriding styles easier.
* **Semantic Naming**: Name IDs and classes based on **purpose** (e.g., .error-message, not .red-text).

**Summary: Use IDs for unique, one-off elements requiring precise targeting, and classes for reusable styles and shared components. Prioritize classes for CSS to maintain flexibility and avoid specificity headaches.**

**P[draggable]{**

**Color: red**

**}**

**Here, p is the html element and draggable is the attribute**

**Index.html**

**<p draggable=”true”>Drag me</p>**

**<p draggable=”false”>Don’t drag me</p>**

**<p draggable=”false”>Don’t drag me</p>**

**Styles.css**

**P[draggable =”false”]{**

**Color: red;**

**}**

**Universal Selector**

Select all

**\*{**

**Color: red**

**}**

**It is universal and when you apply this, it doesn’t matter which class you’ve got, what ID, what attributes set, which different elements,**

**If you select all. It’s going to apply the style to everything where the stylesheet is active.**

**In CSS (Cascading Style Sheets), colors are used to define the visual appearance of elements on a web page, such as text color, background color, borders, and more. There are several ways to specify colors in CSS:**

**### 1. \*\*Color Names\*\***

**CSS supports predefined color names like `red`, `blue`, `green`, etc. There are 140 standard color names supported by modern browsers.**

**```css**

**color: red;**

**background-color: lightblue;**

**```**

**### 2. \*\*Hexadecimal (Hex) Codes\*\***

**Hexadecimal codes are a six-digit representation of RGB (Red, Green, Blue) values. Each pair of digits represents the intensity of red, green, and blue, respectively, ranging from `00` (no intensity) to `FF` (full intensity).**

**```css**

**color: #FF5733; /\* A shade of orange \*/**

**background-color: #33FF57; /\* A shade of green \*/**

**```**

**You can also use shorthand hex codes with three digits, where each digit is a shorthand for two identical digits:**

**```css**

**color: #F53; /\* Equivalent to #FF5533 \*/**

**background-color: #3F5; /\* Equivalent to #33FF55 \*/**

**```**

**### 3. \*\*RGB and RGBA\*\***

**The `rgb()` function allows you to specify colors using decimal values for red, green, and blue, ranging from `0` to `255`.**

**```css**

**color: rgb(255, 87, 51); /\* A shade of orange \*/**

**background-color: rgb(51, 255, 87); /\* A shade of green \*/**

**```**

**The `rgba()` function extends `rgb()` by adding an alpha channel for transparency, where `0` is fully transparent and `1` is fully opaque.**

**```css**

**color: rgba(255, 87, 51, 0.5); /\* Semi-transparent orange \*/**

**background-color: rgba(51, 255, 87, 0.8); /\* Semi-transparent green \*/**

**```**

**### 4. \*\*HSL and HSLA\*\***

**The `hsl()` function specifies colors using Hue, Saturation, and Lightness.**

**- \*\*Hue\*\* is a degree on the color wheel (0-360).**

**- \*\*Saturation\*\* is a percentage (0% is grayscale, 100% is full color).**

**- \*\*Lightness\*\* is also a percentage (0% is black, 100% is white).**

**```css**

**color: hsl(120, 100%, 50%); /\* A bright green \*/**

**background-color: hsl(240, 100%, 50%); /\* A bright blue \*/**

**```**

**The `hsla()` function adds an alpha channel for transparency, similar to `rgba()`.**

**```css**

**color: hsla(120, 100%, 50%, 0.5); /\* Semi-transparent green \*/**

**background-color: hsla(240, 100%, 50%, 0.8); /\* Semi-transparent blue \*/**

**```**

**5. CurrentColor**

**The `currentColor` keyword refers to the current value of the `color` property of the element. It can be useful when you want to apply the same color to multiple properties.**

**```css**

**color: blue;**

**border-color: currentColor; /\* The border will be blue \*/**

**```**

**6. Transparent**

**The `transparent` keyword represents a fully transparent color, which is equivalent to `rgba(0, 0, 0, 0)`.**

**```css**

**background-color: transparent;**

**```**

**7. Gradients**

**CSS also supports gradients, which allow you to create smooth transitions between two or more colors. Gradients are treated as background images.**

**Linear Gradient:**

**```css**

**background: linear-gradient(to right, red, yellow);**

**```**

**Radial Gradient:**

**```css**

**background: radial-gradient(circle, red, yellow);**

**```**

**8. System Colors**

**CSS also provides system colors that match the user's operating system theme. These are less commonly used but can be useful for accessibility.**

**```css**

**color: ButtonText; /\* Text color for buttons \*/**

**background-color: Canvas; /\* Background color for canvas areas \*/**

**```**

**9. Opacity**

**While not a color itself, the `opacity` property can be used to make an entire element (including its content) more or less transparent. The value ranges from `0` (completely transparent) to `1` (fully opaque).**

**```css**

**opacity: 0.5; /\* Makes the element semi-transparent \*/**

**Summary of Color Formats:**

**- \*\*Color Names\*\*: `red`, `blue`, `green`, etc.**

**- \*\*Hexadecimal\*\*: `#RRGGBB` or `#RGB`**

**- \*\*RGB/RGBA\*\*: `rgb(255, 0, 0)` or `rgba(255, 0, 0, 0.5)`**

**- \*\*HSL/HSLA\*\*: `hsl(120, 100%, 50%)` or `hsla(120, 100%, 50%, 0.5)`**

**- \*\*Transparent\*\*: `transparent`**

**- \*\*Gradients\*\*: `linear-gradient()`, `radial-gradient()`**

**- \*\*System Colors\*\*: `ButtonText`, `Canvas`, etc.**

**Each method has its own use case, and the choice depends on the specific needs of your design and how you want to manage colors across your project.**

**Font Properties**

**h1 {**

**colour: blue**

**font-weight: bold**

**1px(pixel) = 1/96th inch(0.26mm) or**

**1pt(point) = 1/72 inch (0.35 mm)**

**font-size: 20px**

**font-family: sans-serif**

**}**

**Font-size: 1px = 1/96th inch(0.26mm)**

**1pt = 1/72 inch(0.35mm)**

**1em (100% of parent) = this means if the body is 20px then h1 is if given 1em then, 1em == 20px and 2em would be 2\*20px**

<body> 20px

<h1>Hello</h1>

</body>

1em =20px & 2em = 2 \* 20px

1rem = closely related to em or the em is the rem, it’s got the extra word, “r”. Basically, its work the same thing. It’s a relative size, but it’s never relative, instead of to the parent or root i.e (<html></html>). So advice to use rem instead of em because em has affect in body where as rem has effect in main root i.e html so it would be less likey to change the root then body.

Font -family is basically the font

h1{

　Font-family: Helvetica, sans-serif

For words with spaces we put inside the quote like “Times New Roman”

}

h2{

　Font-family: “Times New Roman”, serif

}

Fonts.google.com to get more fonts

Choose the fonts and click get fonts and click embed and copy html and CSS as style.

**h1 {**

**text-align: centre or left or right;**

**}**

**We can put our text to centre or left or right using text-align.**

**CSS Box Model- Margin, Padding & Boarder:**

**Each element is a box in itself, and we can change the dimensions of those boxes by changing things such as width & height. The height and width will be measure by either pixels or percentage. And for the box border we need three requirement and they are thickness(px) space solid(style) space and colour:**

**Border: 10px solid black**

colour

Style

Thickness

Up or 1

Right or 2

Left or 4

Down or 3

**For the pixel, if the pixel increases the boarder thickness will increases i.e. the border goes outwards rather than inwards, however, the height and the width of the box won’t change at all.**

**Border properties:**

**Border-top: 0px -> this will set the upper part of border thickness to 0.**

**Border-width: 0px 10px 20px 30px -> this will set the thickness of each side with the corresponding pixel i.e. top would be 0px, right would be 10 px, bottom would be 20px and left would be 30px.**

**NOTE; the sides pixel or border would be in clock wise directions. So when**

**Border-width: 0px 20px is given then up and down would be 0px and left and right would be 20 px or 1+3 = 0px and 2+4 = 20px.**

**Padding:**

**Padding: 20px -> this padding will expand the border with 20px however, the height and width of elements or content will remain same just the broader will get pushed and this expanding will be inside and also there is another called margin: 10px which is the outside of the border which will be any content on the screen. Also, when two borders with the margin of 10px would be the total of 20px separation to each other.**

Margin:10px

padding

elements

Padding margin and border-width are similar like they follow the pixel or sides in clockwise count and for the padding and margin each side are individuals but for border the top and button are corresponding to each other and like wise for left and right but, for margin and padding all sides are free and do not corresponds for example

If we put the margin of button to 10px the content will go 10px further below and for padding as well if we put the padding to 10px in up then the image will further go away but this time up so, we must be careful while distancing the margin and padding.

We can wrap the elements in single box like boxes inside the box using the html tag called <div></div> also known as Content Division Element.

<div></div> are completely invisible unless we apply CSS style to it. And the whole purpose is to act as an invisible box that can contain content in between the opening and the closing tags.

<div>**Content**</div>

<div>**Content**</div>

**The cascade- Specificity and Inheritance**

index.html

<ol>

  <li> One </li>

  <li style="color: green;">Two</li>

  <li>Three</li>

</ol>

style.css

li{

  color: green;

}

<! -- in this above we got three list and CSS colour green so all one, two and three

will be in green colour however, when internal CSS is applying like in list

"two", the colour will be red. all praise to cascade. so, what happening is that,

first the external CSS will run put all the list in green colour but when there is internal CSS

got applied then that internal CSS will override the external CSS like in li "two" the green will be red for it. -->

**There are four broad categories which we look at when we're determining the overall level of importance of CSS rule and these categories are**

**Position:**

li{

  color: red;

  color: blue;

}

li{color: green}

**here in CSS rule the colour will applied per each step down i.e. the ultimate colour applied to green overriding to red and blue. This is known as position rule.**

**so, the lower down the rule is more important, it is.**

**Specificity:**

**CSS specificity refers to how specific a selector is in terms of the elements that you're applying**

**the order would be elements check then it will be overridden by class and class would be overridden with attribute**

**and finally, the id would overridden the attribute so if we place all in line the colour of id would be applied among rest.**

li{color:blue} /\*selecting element 4 \* /

.first-class{color:red} /\*selecting class 3 \*/

li[draggable]{color:purple;} /\*selecting attribute 2 \*/

#first-id {color:orange} /\*final selector id  1 \*/

/\* <li id ="first-id" class="first-class" draggable> \*/

**Type:**

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

<style></style>

<h1 style=" ">hello</h1>

we know that there is 3 type of applying CSS to a particular file.

external CSS by linking to particular file in project folder

another one is using style element with opening and closing tag.

and finally, inline style where the CSS is applied through the style attribute inside the opening

tag of an actual HTML element.

NOW in terms of the cascade, the most important are the styles that are applied inline because this target only

one specific element and here it's just h1.

doesn't matter if we have other 10s h1 it will only apply to this particular element.

Next one would be inline cause this would work only in single webpages and finally the external with the last order so the order would

be external CSS -> 3

internal CSS -> 2

inline CSS -> 1

so, if we apply or point to the same or they all target the same element then the inline CSS would be applied among

both these two external and internal CSS

**Important:**

  color: red;

  color: green !important;

**the final there is keyword called important keyword where we can apply to any CSS rule and this would be the most important and would be applied no matter where it has been applied whether it would be in position or specificity or type.**

Colour would be green not blue

<h1> class="a-class" id="an-id"</h1>

.a-class{

  color: blue;

}

#an-id{

  color: green;

}

**so here is the h1 elements with two lining of class selector with colour blue and**

**an id selector with the colour of green so for importance order**

**the final colour would be an of ID because id is the top in the cascade.**

<h1 class="a-class another-class">hello</h1>

.a-class{

  color: green;

}

.another-class{

  color: blue;

}

**we put multiple class in single element by giving space between classes**

**so in this h1 element the final colour would be blue because this follows the CSS rule of position where one is override as stepping down.**

<html lang="en">

  <head>

    <style>#an-id{color:green}</style>

  </head>

  <body>

    <h1 id="an-id" style="color:blue">Hello</h1>

  </body>

</html>

<! -- here in this case the cascade rule goes from the priority from top to down so even though

we have the style with class selector which falls in specificity however if we look at the cascade rule

type would be much important than specificity so, the <h1> with id and style colour would get matched</h1>

now again if we look at the type the flow priority goes with link first at lowest and style in mid and finally

incline CSS would topped others. so, the final colour would be blue. -->

**CSS Gradient Properties: linear-gradient(), radial-gradient():**

**CSS gradients allow you to create smooth color transitions across elements. There are two main types of gradients in CSS:**

1. **Linear Gradients:**

* **Linear gradients move colors along a straight line.**
* **Syntax: linear-gradient(direction, color1, color2, ...)**
* **The direction can be specified using keywords like to right, to bottom, to top left, etc.**

**Example:**

**background: linear-gradient(to right, blue, green);**

**This will create a horizontal gradient from blue to green.**

1. **Radial Gradients:**

* **Radial gradients expand colors from a central point.**
* **Syntax: radial-gradient(shape size, color1, color2, ...)**
* **The shape can be circle or ellipse.**
* **The size can be a fixed value (e.g., 50px) or a keyword like closest-side.**

**Example:**

**background: radial-gradient(circle, red, yellow);**

**This will create a circular gradient from red to yellow.**

**Gradients are treated as background images in CSS, so they can be used with the background or background-image properties.**

**Some key points about CSS gradients:**

* **They provide a smooth transition between two or more colors.**
* **They can be used to create visually appealing backgrounds and effects.**
* **Gradients are highly customizable, allowing you to control the direction, shape, and color stops.**
* **They are a powerful tool for creating dynamic and engaging visual designs on the web.**

**A pseudo-class can be attached to any selector. It is always written as a colon : followed by a name. For example p:hover.**

**p:hover {**

**background-color: lime;**

**}**

**In the above code, whenever the mouse hovers over a paragraph element, that paragraph will have a lime-colored background.**

**CSS Selector:**

parent

child

selector > selector {

    color: firebrick;

}

Suppose we have div inside div which means nested div.

So outer div will be parent and inner div will be child and in order to give colour or changes to child selector then we can go selector(parent) > selector(child) {color: firebrick}. And this is applied directly to only child not grand or others.

**Descendent:**

descender

descendent

Selector selector {

    color: firebrick;

}

Now here suppose we got two paragraphs inside the div and we use the descendent selector then the parent will show the changes to all the child selector no matter what unlike parent > child. Which means the color will changes to all the selector inside the dive if given as

Div p {

Color :red;

}

**Chaining selector:**

slectorselector {

    color: seagreen;

}

**In chaining selector all the element, id and class and other are chained like written without leaving spaces. This is use for specific targeting the elements.**

**Combining Combiners**

selector selectorselector {

    font-size: 2rem;

}

**Combining the combiners will let us to combine the parent as descendor and child as selectorselectore here as chaining and also other like parent and child relation so that we change to the specific things .**

**CSS Position:**

**Position are Relative, Absolute, Fixed and Static Positioning**

**Static**

**Fixed**

**Absolute**

**Relative**

browser

browser

browser

browserr

**Static Positioning:**

**As soon as we insert any pieces of HTML by default, they’re going to have this particular positioning (Static) applied to them. And this basically means that each item flows as they would in our HTML.**

**Link this site: appbrewery.github.io/css-positioning/**

**Relative Positioning:**

**Item gets positioned relative to its default position.**

**In static positioning, HTML just makes out items go one on top**

**Of the other with almost no space in between unless there is some sort of default.**

**Well, relative positioning, basically takes that original position and you can apply some changes to so you can move it relative to where it should be. Where the blue box is, where it should be and where the blue boxes, it’s been shifted to be pushed from the let and from the top.**

**So that gets this new relative position.**

**Absolute Positioning:**

**Position relative to nearest positioned ancestor or top left corner of webpage.**

**This makes the position relative to the nearest positioned.**

**In Absolute positioning would be inside the parent or ancestor other wise it would positioned at the top left of the corner by default.**

**Fixed positioning:**

**Position would be in constant no matter how the hovering works. Like scrolling the pages from top to down but the position would be fixed and won’t change at all.**