Class 7 - Specificity , Inheritance and Rendering

## Inheritance:

In CSS, certain styles (like font family or color) are like genetic traits; if you set them on a parent element (like body), many of those styles will naturally pass down to the child elements (like paragraphs, divs, etc.), unless the child has its own specific style that overrides it.

## Specificity:

1. order becomes -> !important > inline styles > id > class > element
2. The other way to look at this is - the more specific you are with the rule, the more it takes precedence over the others
    a. Giving a style for divs is generic to all div
    b. Giving a style with a div with a class say “special” is more specific
    c. Giving a style using id is even more specific as there is only one element with that id
    d. Giving an inline style is all the more specific because you added the style on that element itself which in a way can be considered as more specific ( rule applied more closer to the element )
    e. !important is trump card

Priority :

Analogy: Priority in CSS is like the last and most recent instruction given to you.n CSS, if two styles have the same specificity, the one that comes last in the code takes priority, just like the last advice you received.

Bringing It All Together:
When you write CSS, you're giving instructions on how to style your web page. Sometimes these instructions conflict, and that's where these concepts come into play:

Inheritance means that unless you specify otherwise, child elements will use the styles of their parent elements.
Specificity is the system that decides which styles are more "important" based on whether they are defined by a tag name, a class, or an ID.
Priority is about the order of the CSS rules. If two rules have the same specificity, the last one written will be the one applied.

User agent style sheet recap

a. User Agent Style Sheet
b. The user agent stylesheet in CSS is a set of default styles that web browsers apply to HTML documents when no

specific styles are provided by the author. Each web browser has its own user agent stylesheet, and it serves as a baseline style for rendering HTML elements. The term "user agent" refers to the software (web browser) that acts on behalf of the user when interacting with web content.

c.

d. When a browser encounters an HTML document without any external or internal styles (through style sheets), it uses its default styles from the user agent stylesheet to present the content in a readable and visually consistent way. These default styles help ensure a minimum level of readability and usability for web pages.

e.

f. User agent stylesheets define the default rendering of various HTML elements, such as headings, paragraphs, lists, links, and more. The styles include properties like font size, color, margins, padding, and other layout-related attributes. The specific styles can vary between different browsers, reflecting the browser's design choices and preferences.

g.

h. While the user agent stylesheet provides a consistent starting point for rendering HTML content, it's common for web developers to override these styles by applying their own custom styles using external style sheets, internal styles, or inline styles. This allows developers to have more

control over the appearance of their web pages and create a unique design tailored to their needs.

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Sample Page</title>
</head>
<body>
 <h1>This is a Heading</h1>
 <p>This is a paragraph of text.</p>
 <ul>
   <li>Item 1</li>
   <li>Item 2</li>
   <li>Item 3</li>
 </ul>
 <a href="#">This is a link</a>
</body>
</html>
```

i. In this example, if no custom styles are applied, the browser will use its user agent stylesheet to render the HTML elements with default styles. Developers can then use CSS to override these defaults and style the content according to their preferences.

### Cascading Nature of CSS - A Summary

1. Top-Down Application: Styles in CSS are applied in the order they are written in the stylesheet. This means a rule written at the

bottom of your CSS file will be applied after all the rules written above it.

2. Inheritance: Some styles naturally cascade from parent elements to their children. For example, if you set the font-family on the body element, this style will be inherited by all the text inside the body, unless there's a more specific rule applied to the text.
3. Specificity: When two styles conflict, the one with greater specificity takes precedence. Specificity is determined by the type of selector used (like tag, class, ID, or inline style), with each having a different level of "weight" or influence.
4. Last Rule Wins: If two rules have the same level of specificity, the last one written in the CSS file takes precedence.

## Understanding specificity, selector specificity and its values duration

1. First: Create an HTML file to understand specificity. Second: Create an unordered list of let's say "fruits".

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Document</title>
 </head>
 <body>
   <ul>
     <li>Apple</li>
     <li>Mango</li>
     <li>Orange</li>
   </ul>
```

```
</body>
</html>
```

2. Assign 'id' and 'class' to your list.

```
<ul >
     <li>Apple</li>
     <li id="fruits" class ="favourite">Mango</li>
     <li>Orange</li>
   </ul>
```

3. Add the styles

```
<title>Document</title>
   <style>
     .favourite {
       color: green;
     }
     #fruits {
       color: red;
     }
   </style>
```

4. Now the mango turns red because id has more specificity
5. Hover over the rule and see the specificity values
6. What if we add an inline style to this item

```
     <li id="fruits" class="favourite" style="color:
yellow">Mango</li>
```

```
<ul id="fruits">
    <li>Apple</li>
```

```
    <li  class="favourite" >Mango</li>
    <li>Orange</li>
  </ul>
```

a. Now the mango is green again
b. Because the Mango is overriding the inherited values from ul

## Calculating specificity

1. Lets change the html a bit and add another ul
2. Make all li under ul as red

```
<style>
    /* .favourite {
      color: green;
    }
    #fruits {
      color: red;
    } */
    ul li {
      color: red;
    }
  </style>
</head>
<body>
  <ul id="fruits">
    <li>Apple</li>
    <li class="favourite">Mango</li>
    <li>Orange</li>
  </ul>
  <ul>
```

```
    <li>eat</li>
    <li>sleep</li>
    <li>repeat</li>
  </ul>
```

3. If I want to make all the list items for my fruits as red

```
ul#fruits li {
      color: red;
    }
```

4. If I want to make my mangoes as yellow

```
ul#fruits li {
      color: red;
    }

    ul#fruits li.favourite {
      color: yellowgreen;
    }
```

5. Hover over the selector ul#fruits li you will see "Selector Specificity: (1,0,2)".

How to count specificity?

1. The selector ul#fruits li has "Selector Specificity: (1,0,2)".
2. There is one ID i.e., fruits. There are no classes added in the selector. There are 2 elements in the selector: ul and li. Therefore the value will be (1,0,2) (ID , Classes , Elements)
3. Similarly, the value of the selector ul#fruits li.favourite is (1,1,2).

4. How does the Selector Specificity make sure what selector should be applied?
5. By comparing both the Selector Specificity values box by box: we see the value of the selector ul#fruits li.favourite is (1,1,2) which means it has 1 class while the other selector has no class.
6. Therefore, the ul#fruits li.favourite selector will be applied.
1. https://specificity.keegan.st/ online tool to calculate the specificity

Keyword - !important

1. Using !important overrides all other css rules

```
ul#fruits li {
  color: red !important;
}

ul#fruits li.favourite {
  color: yellowgreen;
}
```

## Internal / embedded styles vs external styles

1. We know that we can write css in separate files and link them to our html
2. What is the priority order here ?
3. within the same level of specificity, the order in which the styles are defined or loaded determines which style is applied

4. If two selectors apply to the same element and have the same specificity, the latter one takes precedenc
5. If I have my external css loading after internal then the latter takes precedence

```
<style>
   h1{
     color: red;
   }
   </style>
   <link rel="stylesheet" href="style.css">
</head>
<body>
 <h1>This is a Heading</h1>
```

CSS Inheritance - some nuanced css discussion

1. We will be looking at the four properties - Default, Inherit, Initial and Unset.
2. Default
    a. Default:
    b. Think of Default as the basic setting. It's what the browser automatically uses if you don't tell it to do something else. Each CSS property has a standard, or 'default' setting that it starts with.
    c. Example: When you create a new webpage, the default font size is usually 16px if you don't set it to something else.

d.

3. Inherit
    a. Inherit:
    b. Inherit is like family traits. HTML element can inherit CSS properties from its parent element. When you set a property to 'inherit', you're telling it to use the same value as its parent element.
    c. Example: If the parent element has a font size of 20px, and you set a child element's font size to 'inherit', the child will also have a font size of 20px.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <title>Document</title>
   <style>
       body { font-family: Arial; }
       p{font-family:'Courier New', Courier,
monospace}
       p.special { font-family: inherit; }
     </style>
</head>
<body>
   hello world
   <p>Lorem ipsum, dolor sit amet consectetur
adipisicing elit. Ex, deleniti.</p>
```

```
   <p class="special">This paragraph inherits Arial
font from the body.</p>
   <p>Lorem ipsum dolor sit amet, consectetur
adipisicing elit. Atque nemo dolorum illum fugiat,
doloremque incidunt in quam nobis neque cum.</p>

</body>
</html>
```

4. Initial
    a. Initial is like hitting the reset button. It changes a property back to its original setting, the way it was before any changes were made ( CSS specifications ).
    b. Example: If you've changed the color of text in a paragraph but then decide you want the standard color back, you can set its color to 'initial', and it will revert to the default text color.

```
<!-- intial property-->
   <a href="#">This link is red.</a>
   <a href="#" class="default-color">This link reverts
to default color.</a>
```

    c. styles

```

```

    d. We have styled links (<a>) in our document to be a certain color, say red, but want one specific link to have the

browser's default link color, you can set color: initial; for that link.

5. Unset
    a. It removes any specific instructions you gave for a property, allowing the browser to decide what to do based on the normal rules of CSS. If the property is normally inherited, it will be inherited; if not, it will go back to its default.
    b. Example: If you've set a specific margin for all paragraphs but want one paragraph to just follow the normal rules (like the rest of the document), you can set its margin to 'unset'.
    c. Or, in other words the Unset value removes any specific value set for a property. **If the property is inherited by default (like font-family), it becomes 'inherit'. If not, it reverts to its initial value**
    d. Lets add a margin to all p tags and use unset property to see this behavior

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
   <title>Document</title>
   <style>
     body {
       font-family: Arial;
     }
```

```
    p {
      font-family: "Courier New", Courier, monospace;
      margin: 20px;
    }
    .no-specific-margin { margin: unset; font-family: unset;}

    p.special {
      font-family: inherit;
    }
    a {
      color: red;
    }
    .default-color {
      color: initial;
    }
  </style>
 </head>
 <body>
  hello world
  <p>
    Lorem ipsum, dolor sit amet consectetur adipisicing elit.
Ex, deleniti.
  </p>
  <p class="special">This paragraph inherits Arial font from
the body.</p>
  <p>
    Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Atque nemo
    dolorum illum fugiat, doloremque incidunt in quam nobis
neque cum.
  </p>
  <!-- intial property-->
```

```
    <a href="#">This link is red.</a>
    <a href="#" class="default-color">This link reverts to
default color.</a>
    <!-- unset property -->
    <p>This paragraph has a 20px margin.</p>
<p class="no-specific-margin">This paragraph has an unset
margin.</p>
 </body>
</html>
```

e. What we see is that since margin is not an inhertiable property, it goes to browser's default style values ( no margin ) and since font-family is an inheritable property, it takes the value from the body tag

## CSS Units

1. CSS units are used to specify sizes of various elements and properties in a webpage. Let's go through the most common ones:
2. Pixels (px):
    a. Pixels are like the tiny dots on your screen. When you use pixels, you're saying exactly how many dots something should be tall or wide.
    b. Example: font-size: 16px; makes the text size equivalent to 16 dots high.

c. Use Case: Pixels are great for when you want precise control, like for a border width or a button size.

```
<div class="px-example">Pixel Example</div>
```

d. Add style

```
<style>
    /* Pixels (px) */
    .px-example {
      width: 200px;
      height: 150px;
      font-size: 16px;
      background-color: lightblue;
    }
  </style>
```

3. Percentages
    a. Percentages are relative to something else. For example, if you set a width in percentage, it's a percentage of the width of the parent element.
    b. Example: width: 50%; means the element will take up half the width of its parent.
    c. Use Case: Percentages are useful for responsive design, where you want elements to change size based on the screen or parent element.
    d. Code - add a div in the file

```
<body>
  <div class="px-example">Pixel Example</div>
  <div class="container">
```

```
    <div class="sidebar">Sidebar</div>
    <div class="main-content">Main Content</div>
  </div>
</body>
```

e. Add styles

```
.container {
      display: flex;
      width: 100%; /* Full width of the
parent/container */
    }

    .sidebar {
      width: 25%; /* Takes up 25% of the container's
width */
      background-color: lightcoral;
    }

    .main-content {
      width: 75%; /* Takes up the remaining 75% of
the container's width */
      background-color: lightgreen;
    }
```

f. When using percentages in CSS, the base value for the calculation depends on the property being set ( discuss on ipad ) .

g. Font Size:

   i. When you set the font size in percent, it's based on the font size of the element's parent.

ii. Example: If the parent element has a font size of 16px, and you set a child element's font size to 150%, the child's font size will be 150% of 16px, which is 24px.

h. Padding and Margin:

i. For padding and margin set in percent, the calculation is based on the width of the containing block (the parent element), regardless of whether you're setting vertical or horizontal padding/margin.

ii. Example: If a div has a width of 500px, setting padding: 10%; will result in a padding of 50px on all sides, because 10% of 500px is 50px.

iii. Generally we prefer to give fixed values because it might not be very intuitive otherwise

iv. Try giving padding and margin for sidebar in percent and see that it takes container's width as base value even though we have width defined for it

```
.container {
  display: flex;
  width: 400px; /* Full width of the parent/container */
}

.sidebar {
  width: 25%; /* Takes up 25% of the container's width */
  background-color: lightcoral;
  padding:10%;
  margin:10%;
}
```

i. Width and Height:

        i. Width and height percentages are relative to the dimensions of the parent element.
        ii. Example: If a parent element is 600px wide, setting a child element's width to 50% will make it 300px wide.
    j. Positioning (top, bottom, left, right) ( extra should try ):
        i. When using position: absolute; or position: relative;, the percentages for top, bottom, left, and right are relative to the nearest positioned ancestor's width for left/right and height for top/bottom.
        ii. Example: If a positioned div is inside a parent div that's 400px wide, setting left: 20%; on the child div would move it 80px (20% of 400px) from the left edge of the parent.
    k. Line Height ( extra can try ) :
        i. When setting line height in percent, it's relative to the element's own font size.
        ii. Example: If an element has a font size of 20px, setting line-height: 150%; results in a line height of 30px.
4. Ems
    a. For ems, behavior changes depending on the context in which they're used
    b. For Font Sizes:
        i. When em is used for font sizes, it is relative to the font size of the parent element.

ii. Example: If the parent element has a font size of 16px, and you set a child element's font size to 1.5em, the font size of the child will be 24px (1.5 times 16px).

c. For Other Properties (like Padding, Margin, Width, etc.):
   i. When em is used for other properties like padding, margin, or width, it's relative to the font size of the element itself, not the parent.
   ii. Example: If an element has a font size of 20px, and you set its padding to 2em, the padding will be 40px (2 times 20px), regardless of the parent's font size.

d. Create a container for em and two child elements

```
<div class="em-container">
     <p class="text">This is a paragraph with standard text.</p>
     <div class="box">Content inside the box.</div>
   </div>
```

e. Add styles

```
.em-container {
       font-size: 20px; /* Base font size for the container */
     }

     .text {
       font-size: 0.8em; /* Smaller than the container font size */
     }
```

```
.box {
  font-size: 1.5em; /* Larger than the container font size */
  padding: 1em;
  border: 1em solid black; /* Border width also relative to the container's font size */
}
```

f. Notice that font size are relative to parent's and padding and border are related to current element's font size
g. Compounding effect issue
    i. There can be compounding effect with ems units and can quickly grow or decrease in size when nested in unexpected way
    ii. html

```
<div class="grandparent">
  Grandparent Text
  <div class="parent">
    Parent Text
    <div class="child">Child Text</div>
  </div>
</div>
```

    iii. css

```
.grandparent {
  font-size: 16px; /* Base font size */
}
```

```
.parent {
  font-size: 1.5em; /* 150% of 16px = 24px
*/
}

.child {
  font-size: 1.5em; /* 150% of 24px
(parent's size), not 16px */
}
```

5. Rems
    a. Rems are like ems, but they always refer to the base font size of the document ( font size of the html tag ), not the font size of the parent element.
    b. Examples: If the base font size is 16px, font-size: 2rem; would be 32px.
    c. Use Case: Rems are great for consistent scaling across your website, as they always refer to a single base size.
    d. style

```
html{
        font-size: 22px;
    }
```

```
.rem-example {
    margin: 1rem; /* Always equivalent to 16px */
    background-color: lightgoldenrodyellow;
  }
```

    e. html

```
    <div class="rem-example">REM Example</div>
```

g.

h.

6. Viewport Width (vw) and Viewport Height (vh)
    a. These are based on the size of the browser window. 1vw is 1% of the viewport's width, and 1vh is 1% of the viewport's height.
    b. Examples: width: 50vw; would make an element half as wide as the browser window. height: 100vh; would make an element as tall as the whole viewport.
    c. Use Case: Viewport units are excellent for creating responsive designs that adjust to different screen sizes.
    d. code

        Viewport Example
    e. css

```
.viewport-example {
    width: 50vw;
    height: 30vh;
    background-color: lightcoral;
  }
```