Class-7 CSS Cascading, Specificity, Inheritance and Positioning

---

title: Agenda description: duration: 220 card_type: cue_card

The topics we are going to cover in today's session are:

- Cascading in CSS and how it works
- Specificity in CSS
- Inheritance
- Positioning in CSS

---

title: What is Cascading in CSS? description: Understanding the definition of "Cascading" in Cascading Style Sheet duration: 200 card_type: cue_card

What is Cascading in CSS?

Before Directly Jumping to Cascading Explain about User agent Stylesheets

User Agent Style Sheet

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.

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.

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.

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.

Here's a brief example of how user agent styles might be applied to an HTML document:

```html
<!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>
```

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. In layman's terms, the "cascading" in CSS can be likened to a waterfall flowing down a series of steps. Just as water flows from the top and cascades down, styles in CSS are applied from the top of the stylesheet to the bottom, with later styles flowing down and potentially overriding earlier ones.

2. 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.

3. 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.

4. Last Rule Wins: If two rules have the same level of specificity, the last one written in the CSS file takes precedence. This is like the water at the bottom of the waterfall being the most recent to fall, and it's what determines the final outcome. Understanding these principles helps you predict how styles will be applied and how to effectively structure your stylesheets. It's the cascading nature of CSS that gives it both its power and its elegance.

5. 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.

First , let's take up Specificity and explore it in deatail that how diffrent selectors have different weightage

---

title: Specificity description: Understanding specificity, selector specificity and its values duration: 930 card_type: cue_card

Let's Explore Specificity

Specificity

To understand specificity let's take a real life Example.

Real-life analogy: The Military Decree System

- Analogy: Specificity can be compared to a hierarchy or rank in a game or military. Imagine a group of knights, archers, and soldiers. A general's command (an ID in CSS) is followed over a captain's (a class in CSS), and a captain's command is followed over a regular soldier's (an element or tag in CSS). If a soldier is given an order from both a captain and a general, they will follow the general because the general has higher rank or specificity.

- Using the same analogy, it happens sometimes that certain individuals or soldiers are chosen to do a covet operations right from the highest authority. These are like inline styling that takes precedence over other styling be it from element, class or id

- But there is a trump card - !important For now just understand that this is like a royal decreeThe Royal Edict (!important): At the top is the king or queen's royal edict. When they stamp a decree with the royal seal (!important), it must be followed above all else, no matter what the generals, captains, or soldiers have decided. This is the most powerful and overrides all other orders.

- But use it sparingly; just as a kingdom where the ruler makes too many overriding decrees can become chaotic, overusing !important can make your stylesheets difficult to maintain and debug. It's best reserved for special cases where you need to enforce a style that absolutely must take precedence."

Final order becomes -> !important > inline styles > id > class > element

Now let's understand this in CSS terms-

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>
```

- Now add id and Class as shown below

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

- Add styles to both class and the id

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

```
    }
  </style>
```

Now the mango turns red because id has more specificity Hover over the rule and see the specificity values

What if we add an inline style to this item

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

Now the Inline style takes Precedence

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>
```

Fourth: Using CSS, make the colour of the class "favourite" blue.

The Selector will be id i.e., "fruits" Inside this id select a list element and then a specific class i.e., "favourite".

```
<style>
    ul#fruits li.favourite{
      color: blue;
    }
</style>
```

Fifth: Using CSS, change the colour of the unordered list to blue as well.

```
<style>
    ul#fruits li.favourite{
      color: blue;
    }

    ul#fruits li{
        color: blue;
    }
</style>
```

[Ask the learners]

What will happen if we change the colour of the unordered list to blue after changing the colour of class "favourite" to red?

```
<style>
    ul#fruits li.favourite{
      color: red;
    }

    ul#fruits li{
        color: blue;
    }
</style>
```

–> The result will be that **"Mango" will be of red** colour and the rest list elements will be blue because it is using the **Specificity** property. Hover over the selector `ul#fruits li` you will see "Selector Specificity: (1,0,2)".

Before understanding the Selector Specificity values:

[Ask the learners]

What if the style attribute is applied in the list tag with class = "favourite"?

```
<body>
    <ul id = "fruits">
        <li>Apple</li>
        <li class ="favourite" style= "color: yellow;">Mango</li>
        <li>Orange</li>
    </ul>
</body>
```

–> The result will be that **the value of Mango will turn to "yellow"** and the rest of the list elements will remain blue because the style attribute does not get affected by the style tag.

Understanding values in specificity

Specificity can have **four** values if the style attribute is also included.

Style attribute	IDs	Classes	Elements

How to count specificity?

The selector `ul#fruits li` has "Selector Specificity: (1,0,2)". But how?

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)

Similarly, the value of the selector `ul#fruits li.favourite` is (1,1,2).

How does the Selector Specificity make sure what selector should be applied?

> Remove style attribute from the list element with class = "favourite".

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. Therefore, the `ul#fruits li.favourite` selector will be applied.

---

title: practice questions on Specificity description: duration: 630 card_type: cue_card

[Ask the learners]

Arrange them from the least effective to the most effective selector.

1. `.test`
2. `h1.test`
3. `#try`
4. `h1`

–> **4<1<2<3** is the order.

Let's compare the values of each one of them.

1. `.test` - It has one class. Therefore, the value is (0,1,0).
2. `h1.test` - It has one class and one element. Therefore, the value is (0,1,1).
3. `#test` - It has one ID. Therefore, the value is (1,0,0).
4. `h1` - It has one element. Therefore, the value is (0,0,1).

[Ask the learners]

Calculate the value of the Selector Specificity of the following selector. `#try ul div.test h2{}`

–> (0,1,1,3)

ID - #try Class - .test Element - ul, div, h2

Style attribute	ID	Class	Element
0	1	1	3

[Ask the learners]

Calculate the value of the Selector Specificity of the following selector. `#try span img .test .main header`

–> (0,1,2,3)

ID - #try Class - .test, .main Element - span, img, header

Style attribute	ID	Class	Element
0	1	2	3

> You can use Keegan to calculate Specificity value.

---

title: !important keyword description: duration: 150 card_type: cue_card

Keyword - `!important`

If the keyword !important is used then it follows the cascading rule regardless of the Specificity Value. Use this keyword only once in a selector. If it is used twice for the same selector then the Specificity rule will be followed.

Example

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

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

All the list elements will turn into a blue colour regardless of the Specificity Value.

> Priority of Inline CSS will be more than Internal CSS and External CSS.

> The priorities of the External CSS file and Internal CSS file can be changed by following the cascading rule. That means, whatever comes later will be followed.

---

title: CSS Inheritance description: Understanding the properties of inheritance - initial, unset, inherit with example. duration: 1290 card_type: cue_card

CSS Inheritance

Definition

As the last names are inherited in a family same way inheritance works in CSS which inherits some property from the parent.

In CSS, the `inherit` property allows an element to inherit the computed value of a property from its parent element. This means that if a property is set to `inherit` for a specific element, that element will inherit the value of that property from its parent element.

Here's an example to illustrate how the `inherit` property works:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inherit Property Example</title>
  <style>
    /* Define a style for the parent element */
    .parent {
      color: blue; /* Set text color to blue */
      font-size: 20px; /* Set font size to 20 pixels */
    }

    /* Define a style for the child element */
    .child {
      color: inherit; /* Inherit text color from parent */
      font-size: inherit; /* Inherit font size from parent */
    }
  </style>
</head>
<body>
  <!-- Parent element -->
  <div class="parent">
    <!-- Child element -->
    <p class="child">This text will inherit color and font size from its parent.
  </div>
</body>
</html>
```

In this example:

- The `.parent` class sets the text color to blue and the font size to 20 pixels.
- The `.child` class uses the `inherit` value for both `color` and `font-size` properties, which means it will inherit these properties from its parent element (`.parent`).
- So, the text inside the `<p>` element with the class `.child` will appear in blue color and with a font size of 20 pixels, as it inherits these properties from its parent element.

---

title: CSS Positioning description: duration: 1290 card_type: cue_card

CSS positioning

In CSS, the `position` property is used to control the positioning of an element within its containing element. The `position` property can take several values, and one of them is `static`. When an element's position is set to `static`, it means that the element will be positioned according to the normal flow of the document.

Here are some key characteristics of an element with `position: static`:

1. **Default Behavior:** This is the default value for the `position` property. Elements with `position: static` are positioned in the normal flow of the document, meaning they will be displayed one after another, vertically and horizontally, based on their order in the HTML.

2. **No Offsetting:** Elements with `position: static` are not affected by the `top` , `right` , `bottom` , and `left` properties. Any attempts to apply these properties will have no effect.

Here's an example:

```
.normal-flow {
  position: static;
  border: 1px solid #000;
  padding: 10px;
}
```

```
<div class="normal-flow">This is a static positioned element.</div>
```

In this example, the `.normal-flow` element will be displayed in the normal flow of the document, and any attempt to offset it using `top`, `right`, `bottom`, or `left` will have no effect.

```
.normal-flow {
  position: static;
  border: 1px solid #000;
  padding: 10px;
  top: 20px; /* This has no effect */
  left: 30px; /* This has no effect */
}
```

So, in summary, `position: static` is the default positioning behavior, and elements with this position value are placed in the normal flow of the document without any special positioning.

Now how else can Positioning work let's understand

title: Types of Positions? description: This section explains everything about Position Property in CSS duration: 600 card_type: cue_card

So, let's understand two types of position in CSS:

- **Absolute Position**
- **Relative Position**

Think about a table you have in front of you. Imagine you're looking straight down at the table. It's just a flat square surface, like the top of the table. This is the place where you might put things, like your phone or a cup of coffee.

What we're going to talk about is how you can move this "cell phone" around on the table in two different ways.

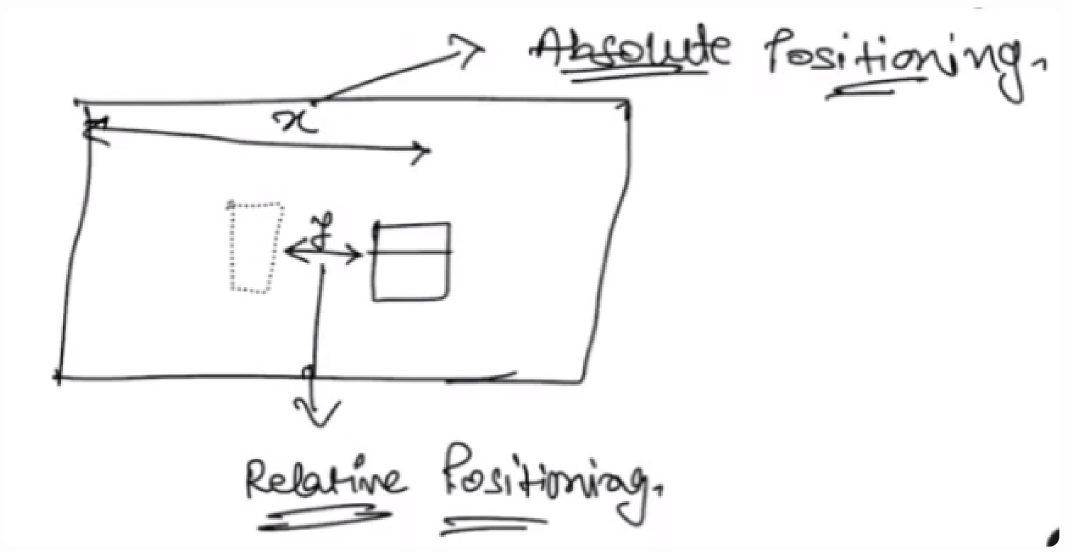
[Ask the learners] Can anyone tell what are the two ways we can move this cell phone on table.

When we talk about moving the box around, we're talking about changing its position in these two main directions – X-Axis and Y-Axis. These two coordinates, X and Y, help us understand exactly where the box (or any object) is located on the table.

Let's say you start with the cell phone placed at a specific spot on the table. Now, you want to move it a bit. In CSS terms, this is done by specifying how many centimeters (like pixels) you want to move the box from its current position.

For example, if you move the box 10 centimeters to the right. So the relative position of the cell phone from its starting point to ending point has changed by 10 cm.

If you are measuring the distance from the starting of the Table. The distance between starting of the table and the cell phone will be the absolute position.



So we can define in terms of CSS as:

- **Relative Position:** The element is positioned relative to its previous position.
- **Absolute Position:** The element is positioned absolutely to its parent container.

Now, that we know what is Relative and Absolute Positioning. Let's move forward by coding and understanding positions.

Position : realtive

In CSS, the `position: relative` property is used to position an element relative to its normal position in the document flow. To explain this with a real-life analogy, let's consider a scenario of arranging books on a shelf.

Imagine you have a bookshelf, and each book represents an HTML element on a webpage. By default, the books are placed on the shelf in a specific order, much like how HTML elements are positioned in the normal document flow. Now, if you decide to move a particular book a few inches to the right or left, you're changing its position relative to its normal place on the shelf.

In CSS, the `position: relative` property allows you to make similar adjustments to the positioning of an element. When you apply `position: relative` to an element, it remains in the normal document flow, but you can then use the `top`, `right`, `bottom`, or `left` properties to move it from its original position.

Here's a simple example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    .bookshelf {
      position: relative;
      width: 300px;
      height: 200px;
      border: 1px solid #ccc;
    }

    .book {
      position: relative;
      width: 50px;
      height: 100px;
      background-color: #3498db;
      color: #fff;
      text-align: center;
      line-height: 100px;
    }

    .book2 {
      position: relative;
      top: 20px;
      left: 30px;
    }
  </style>
</head>
<body>
  <div class="bookshelf">
    <div class="book">Book 1</div>
    <div class="book book2">Book 2</div>
  </div>
</body>
</html>
```

In this example, the `.book2` class has `position: relative`, and it is moved 20 pixels down and 30 pixels to the right from its normal position within the `.bookshelf`. The positioning is relative to where it would have been in the normal flow.

You can experiment with the values of `top`, `right`, `bottom`, and `left` to see how they affect the positioning of the element within its containing element. Keep in mind that using `position: relative` may not cause a visible change unless accompanied by additional positioning properties.

Position : absolute

Let's continue with the bookshelf analogy to explain the `position: absolute` property in CSS.

In our bookshelf scenario, `position: absolute` can be said to taking a book out of the regular flow of the shelf and placing it in a specific position relative to the bookshelf itself. When a book has an absolute position, it's as if it's no longer part of the standard order on the shelf, and you can position it anywhere you want, ignoring the presence of other books.

In CSS, applying `position: absolute` to an element takes it out of the normal document flow and positions it relative to its closest positioned ancestor (an ancestor element that has a position value other than `static`) or, if there is none, relative to the initial containing block, usually the viewport.

Here's an example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    .bookshelf {
      position: relative;
      /* Add and remove this position propery to show how absolute position will
      width: 300px;
      height: 300px;
      border: 1px solid #ccc;
    }

    .book {
        display: inline-block;
    }

    .book1 {
      background-color: #3498db;
      width: 50px;
      height: 100px;
      color: #fff;
      text-align: center;
      line-height: 100px;
      margin: 5px;
    }

    .book2 {
      background-color: red;
      width: 50px;
      height: 100px;
      color: #fff;
      text-align: center;
      line-height: 100px;
      margin: 5px;
      /* Carefully add these properties and explain line by line */

      position: absolute;
      top : 30px;
      left: 50px;

    }

    .book3 {
      background-color: green;
      width: 50px;
      height: 100px;
      color: #fff;
      text-align: center;
      line-height: 100px;
      margin: 5px;

    }

    .book4 {
      background-color: magenta;
      width: 50px;
      height: 100px;
      color: #fff;
      text-align: center;
      line-height: 100px;
      margin: 5px;

    }
  </style>
</head>
<body>
  <div class="bookshelf">
    <div class="book book1">Book 1</div>
    <div class="book book2">Book 2</div>
    <div class="book book3">Book 3</div>
    <div class="book book4">Book 4</div>
  </div>
</body>
</html>
```

In this example, the `.book2` class has `position: absolute`, and it is positioned 20 pixels from the top and 30 pixels from the left of the closest positioned ancestor, which is the `.bookshelf`. Notice that Book 2 is now positioned independently of Book 1 and does not affect the layout of other elements.

In simpler terms:

1. **Normal Flow (Default Order):** Elements are like books on a shelf, placed in a default order.

2. **`position: absolute`:** It's like taking a book out of the normal order and positioning it wherever you want, regardless of where other books are. The book (element) is no longer affecting the order of other books.

In this example, "Book 2" (element with `class="book2"`) is using `position: absolute` to be placed wherever you specify (20px from the top and 30px from the left). It's like moving a book independently on the shelf without affecting others.

Experiment with different values for `top`, `right`, `bottom`, and `left` to observe how the absolute positioning works in relation to its closest positioned ancestor or the initial containing block. It's a powerful tool for precise element placement on a webpage.

Fixed position:

- Fixed Position is basically when your Element will take a place with the Respect to the window and it will not move from there.
- Fixed-positioned element is "fixed" in a specific location on the screen, and it won't move when the user scrolls up or down the page. This can be useful for creating elements that should always be visible, like navigation bars or call-to-action buttons, regardless of where the user is on the page.
- The element will maintain its position relative to the viewport's coordinates, providing a consistent visual reference point as the user interacts with the content.

Exercise 4

Problem Statement

Give **fixed** position value to box 3 and fix it at the bottom of the scrollable page.

Solution

- In style of box_3, we can use position property and set the value as **fixed**.
- For fixing it to the bottom of the page, we can give right property as 4px and bottom as 1px.

Pseudocode

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Positioning</title>
    <style>
        .container {
            background-color: dodgerblue;
            height: 4000px;
        }

        .box {
            display: inline-block;
            height: 150px;
            width: 150px;
            background-color: tomato;
            margin: 10px;
        }

        #box_3 {
            position: fixed;
            right: 4px;
            bottom: 1px;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="box">1</div>
        <div class="box">2</div>
        <div class="box" id="box_3">3</div>
        <div class="box">4</div>
    </div>
</body>

</html>
```

Now if we scroll the page, the box3 will be fixed at the bottom right of the page.

Sticky:

- When an element is given a "position" value of "sticky," it acts like a relative-positioned element within its containing element until a certain scroll threshold is reached. Once the user scrolls beyond that threshold, the element becomes "stuck" in place and behaves like a fixed-positioned element, remaining visible on the screen.
- In other words, a sticky element starts as part of the normal document flow, just like a relatively positioned element. As the user scrolls, the element follows its normal position until it reaches a designated point (usually when its top or bottom edge reaches a specific distance from the viewport's edge). At that point, it becomes "sticky" and remains fixed at that position while the rest of the content scrolls.

Lets go to the zomato website and see its navbar, here you can see when we scroll the page, this navbar is getting fixed at the top of the page.

So on reaching a particular value, sticky gets fixed.

Exercise 5

Problem Statement

Give **sticky** position value to box 3 and fix it at the top of the scrollable page.

Solution

- In style of box_3, we can use position property and set the value as **sticky**.

- For fixing it to the top of the page, we can give top property as 0.

Pseudocode

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Positioning</title>
    <style>
        .container {
            background-color: dodgerblue;
            height: 4000px;
        }

        .box {
            display: inline-block;
            height: 150px;
            width: 150px;
            background-color: tomato;
            margin: 10px;
        }

        #box_3 {
            position: sticky;
            top: 0;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="box">1</div>
        <div class="box">2</div>
        <div class="box" id="box_3">3</div>
        <div class="box">4</div>
    </div>
</body>

</html>
```

Now if we scroll the page, the box3 will behave normally till it touches the top and them it becomes fixed to the top of the page.

That's all for today's session!

That's all for Today's Session