Cascading Style Sheets (CSS)

Unit-III

SCS181 Web Technologies-I

Er. Anu Arora, Assistant Professor, GNA University

Introduction to CSS

Cascading Style Sheets (CSS) is a language used to describe the visual presentation of web pages, including colors, layout, and fonts.

It plays a crucial role in modern web development by providing a way to enhance the appearance of HTML or XML documents.

With CSS, developers can separate the design aspects from the content, making websites more visually appealing and easier to maintain.

Purpose of CSS

CSS is designed to enhance the aesthetics and usability of websites, offering tools for customizing the layout, colors, typography, spacing, and other visual elements. By implementing CSS, web developers can:

- **Control layout and design:** Define how elements are displayed on different screen sizes and devices, ensuring responsive design.
- **Improve content accessibility:** Customize layouts for different platforms (desktops, tablets, smartphones, etc.) or assistive technologies like screen readers.
- **Enhance maintainability:** Use reusable styles across multiple pages by linking a single CSS file, reducing duplication and enabling easier updates.
- **Ensure consistency:** Apply uniform styles to multiple elements and pages, creating a cohesive look and feel across an entire website.

Structure of CSS

- CSS is made up of rules that define how HTML elements should be styled. Each rule consists of a selector and a declaration block.
- The selector identifies which HTML elements the styles will apply to, while the declaration block contains one or more declarations.
- Each declaration consists of a property and a value, separated by a colon (`:`), and the declarations are enclosed in curly braces (`{}`).

Example of a simple CSS rule:

```
p {
    color: blue;
    font-size: 16px;
}
```

In this example:

The selector is `p`, which targets all paragraph elements (`<p>`).

The declaration block contains two declarations:

- `color: blue;` changes the text color to blue.
- `font-size: 16px;` sets the font size to 16 pixels.

The Cascading Nature of CSS

The "Cascading" in Cascading Style Sheets refers to the way styles are applied to elements. CSS follows a set of rules known as the cascade to determine which styles are applied when multiple rules could apply to the same element.

This involves three key concepts:

1. Inheritance: Some CSS properties (like color and font-family) are inherited from a parent element to its children. Others (like margin and padding) are not.

2. Specificity: When multiple styles apply to the same element, CSS uses a system of specificity to determine which rule takes precedence. For example, an ID selector (`#id`) is more specific than a class selector (`.class`), and a class selector is more specific than a type selector (`element`).

3. Source Order: When specificity is the same, the last rule in the source code takes precedence.

Example:

```
<p id="intro" class="text">This is a
paragraph.</p>

p {

    color: green; /* Less specific */

}

#intro {

    color: red; /* More specific */

}

.text {

    color: blue; /* Class-level specificity */

}
```

Key Features of CSS

CSS offers a variety of features to control the appearance of web pages:

- **Typography:** Control fonts, font sizes, weights, styles, and line spacing.
- **Colors:** Define text, background, and border colors using various formats like named colors (`blue`), hexadecimal values (`#0000FF`), RGB (`rgb(0, 0, 255)`), or HSL (`hsl(240, 100%, 50%)`).
- **Box Model:** The box model consists of margins, borders, padding, and the content area. Understanding this is essential for controlling the spacing and layout of elements.
- **Positioning:** CSS provides several positioning schemes such as `static`, `relative`, `absolute`, `fixed`, and `sticky` to control how elements are placed on the page.
- **Flexbox and Grid:** These are powerful layout systems for building responsive and flexible designs. Flexbox is a one-dimensional layout system, while Grid is two-dimensional.
- **Transitions and Animations:** CSS allows smooth transitions between different styles, as well as keyframe animations to create complex effects without the need for JavaScript z

Types of CSS

CSS can be applied to HTML documents in three different ways:

- Inline CSS
- Internal CSS
- External CSS.

Each method has its specific use cases, benefits, and limitations.

Inline CSS

Inline CSS is used to apply styles directly to individual HTML elements using the `style` attribute. The styles are written inside the opening tag of the element.

This method is considered less efficient for larger projects because it mixes content and presentation, reducing maintainability. However, it can be useful for quickly styling a single element or overriding other styles.

Syntax:

```
<element style="property: value;">
```

Internal CSS (Embedded CSS)

Internal CSS is defined within a `<style>` block inside the `<head>` section of an HTML document.

This method is useful when you want to apply styles to a single HTML document or when you don't want to link an external file.

It keeps the CSS code in one place, separate from the HTML content, but within the same file.

Syntax:

```
<style>

  /* CSS rules go here */

</style>
```

External CSS

External CSS is the most commonly used method in modern web development.

The styles are written in a separate `.css` file, and the HTML document links to that file using the `<link>` tag in the `<head>` section.

External CSS is the best practice for large websites or multi-page projects because it allows you to reuse the same stylesheet across multiple pages.

Syntax (in HTML file):

```
<link rel="stylesheet" href="path-to-your-style.css">
```

Comparison of CSS Types

Feature	Inline CSS	Internal CSS	External CSS
Location	Inside HTML elements	<style> block in <head>	Separate .css file linked to the document
Usage	Quick, one-off styles	Page-specific styles	Global styles for multiple pages
Maintainability	Poor	Fair	Excellent
Reusability	None	Limited	High
Specificity	High (overrides other styles)	Medium	Medium
File size	Can make the HTML heavy	Keeps HTML and CSS separate, but still in one file	Keeps HTML light, CSS managed separately

Best Practices for Using CSS Types

1. **Use External CSS for Large Projects:** It is highly recommended to use external stylesheets in large web applications because they offer better maintainability, scalability, and reusability across multiple pages.

2. **Minimize the Use of Inline CSS:** Although inline CSS has the highest specificity and can override other styles, it should be avoided for anything beyond quick, temporary fixes. Inline styles make code harder to read and maintain.

3. **Internal CSS for Small or Single Page Projects:** Internal CSS can be a good option for small projects or single-page websites where you don’t need to manage multiple stylesheets.

4. **Responsive Design:** External CSS is particularly useful when working on responsive design. It allows developers to use media queries and adjust layouts across different screen sizes and devices from a single file.

5. **Avoid Duplication:** Inline CSS and internal CSS can lead to style duplication if used across different elements or pages. External CSS avoids this problem by allowing styles to be reused across the entire website.

CSS Properties

1. Layout Properties

a. `display`: Defines how an element is displayed on the page.

Values:

- `block`: Displays the element as a block (occupies full width).
- `inline`: Displays the element inline (flows with text).
- `none`: Hides the element (it doesn’t occupy space).
- `flex`: Enables Flexbox layout.
- `grid`: Enables Grid layout.

Example:

```
div {
display: block;
}
```

b. `position`: Controls how an element is positioned on the page.

Values:

- `static` (default): Normal flow of the document.
- `relative`: Positioned relative to its normal position.
- `absolute`: Positioned relative to its nearest positioned ancestor.
- `fixed`: Positioned relative to the browser window (doesn’t scroll with content).
- `sticky`: Switches between relative and fixed based on scroll position.

Example:

```
header {
    position: fixed;
    top: 0;
    width: 100%;
}
```

1. Layout Properties

c. `float`: Floats an element to the left or right, allowing text and inline elements to wrap around it.

Values: `left`, `right`, `none`

Example:

```
img {
    float: left;
    margin-right: 10px;
}
```

d. `z-index`: Defines the stacking order of elements (higher values are placed in front of lower ones).

Values: Numeric values (e.g., `1`, `10`, `100`)

Example:

```
.box {
    position: relative;
    z-index: 10;
}
```

1. Layout Properties

e. `overflow`: Specifies what happens if content overflows an element’s box.

Values:

- `visible`: Content is not clipped (default).
- `hidden`: Content is clipped, and the overflow is not visible.
- `scroll`: Content is clipped, but a scrollbar appears.
- `auto`: Scrollbars appear only if the content overflows.

Example:

```
div {
    width: 300px;
    height: 200px;
    overflow: scroll;
}
```

2. Text and Font Properties

a. `color`: Sets the text color of an element.

Values: Color names (e.g., `red`), hexadecimal values (e.g., `#ff0000`), RGB, HSL.

Example:

```
p {
    color: #333;
}
```

b. `font-family`: Specifies the font(s) to be applied to the text. It is best practice to include a fallback font in case the primary font is not available.

Values: Font names (e.g., `Arial`, `Verdana`), generic family names (`serif`, `sans-serif`, `monospace`).

Example:

```
body {
    font-family: 'Helvetica', sans-serif;
}
```

2. Text and Font Properties

c. `font-size`: Sets the size of the text.

Values: Length values (`px`, `em`, `rem`), percentages (`%`), or predefined sizes (`small`, `medium`, `large`).

Example:

```
h1 {
    font-size: 36px;
}
```

d. `font-weight`: Controls the thickness of the text.

Values: `normal`, `bold`, `bolder`, `lighter`, numeric values (e.g., `100`, `400`, `700`).

Example:

```
strong {
    font-weight: bold;
}
```

2. Text and Font Properties

e. `text-align`: Specifies the horizontal alignment of text.

Values: `left`, `right`, `center`, `justify`.

Example:

```
h2 {
    text-align: center;
}
```

f. `text-decoration`: Applies decorations to text, such as underlining or overlining.

Values: `none`, `underline`, `overline`, `line-through`.

Example:

```
a {
    text-decoration: none;
}
```

2. Text and Font Properties

g. `line-height`: Controls the amount of space between lines of text (leading).

Values: Numeric values (`1.5`), length (`px`, `em`), percentage (`%`).

Example:

```
p {
    line-height: 1.5;
}
```

h. `letter-spacing`: Specifies the space between characters.

Values: Length values (e.g., `0.1em`, `2px`).

Example:

```
h1 {
    letter-spacing: 2px;
}
```

3. Box Model Properties

a. `margin`: Sets the space outside an element’s border.

Values: Length (`px`, `em`, `%`), auto (centers the element horizontally), can specify values for each side (`margin-top`, `margin-right`, `margin-bottom`, `margin-left`).

Example:

```
div {
    margin: 20px;}
```

b. `padding`: Defines the space between an element’s content and its border.

Values: Length values (`px`, `em`), percentages, can specify values for each side (`padding-top`, `padding-right`, `padding-bottom`, `padding-left`).

Example:

```
section {
    padding: 10px;
 }
```

3. Box Model Properties

c. `border`: Sets the border around an element.

Values:

Border-width (e.g., `1px`),
Border-style (e.g., `solid`, `dotted`, `dashed`),
Border-color (e.g., `black`, `#ff0000`).

Example:

```
div {

    border: 2px solid black;

  }
```

d. `width` and `height`: Set the width and height of an element.

Values: Length values (`px`, `%`, `em`, `auto`).

Example:

```
 img {

     width: 100%;

     height: auto;

 }
```

4. Background Properties

a. `background-color`: Defines the background color of an element.

Values: Color names (e.g., `red`), hexadecimal (`#ff0000`), RGB, HSL.

Example:

```
 body {
    background-color: #f4f4f4;
 }
```

b. `background-image`: Sets an image as the background of an element.

Values: URL of the image file (e.g., `url('image.jpg')`).

Example:

```
div {
    background-image: 
url('background.png');
  }
```

4. Background Properties

c. `background-position`: Controls the position of the background image.

Values: Keywords (`left`, `right`, `center`), or length values (`px`, `%`).

Example:

```
body {
    background-position: center;
  }
```

d. `background-size`: Specifies the size of the background image.

Values: `cover` (resize to cover the element), `contain` (resize to fit inside), length values (`px`, `%`).

Example:

```
section {
    background-size: cover;
  }
```

5. Flexbox Properties

a. display: flex

Enables a flex container, providing a flexible way to layout and align items.

Example:

```
.container {
   display: flex;
   justify-content: space-
between;
}
```

b. justify-content

Aligns flex items horizontally.

Values: flex-start, flex-end, center, space-between, space-around.

Example:

```
.container {
   justify-content: space-
between;
}
```

5. Flexbox Properties

c. align-items

Aligns flex items vertically within the container.

Values: stretch, flex-start, flex-end, center, baseline.

Example:

```
.container {
   align-items: center;
}
```

6. Grid Properties

a. display: grid

Enables a grid layout.

Example

```
.grid-container {
   display: grid;
   grid-template-columns: 1fr 1fr 1fr;
```