Week 3: Introduction to CSS

CSS (Cascading Style Sheets) is the styling language used to define the appearance of HTML elements on a webpage. This week, we'll dive into the foundational concepts of CSS syntax, selectors, properties, and values.

1. CSS Syntax: Understanding the Structure of CSS Rules

CSS rules are composed of three primary components: selectors, properties, and values. This syntax structure enables us to target specific HTML elements and apply styling to them.

CSS Rule Structure

- **Selector**: Identifies the HTML element(s) to be styled.
- **Property**: Defines the characteristic or attribute to style, like `color` or `font-size`.
- **Value**: Specifies the settings for the property, like `red` for color or `20px` for font size.

Example of CSS Syntax

```
h1 {
  color: blue;
  font-size: 24px;
}
```

In this example:

- `h1` is the **selector** targeting all `<h1>` elements.
- `color` and `font-size` are **properties** being modified.
- `blue` and `24px` are **values** assigned to these properties.

2. CSS Selectors: Targeting HTML Elements with Precision

Selectors are crucial in CSS because they allow us to specify which elements on a page we want to style. There are several types of selectors, each serving a unique purpose.

Types of CSS Selectors

1. **Element Selectors**
    - Targets all instances of a specified HTML element.
    - Useful for setting general styles for common elements like paragraphs (`<p>`) or headings (`<h1>`).

    Example:

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

Here, all `<p>` elements on the page will have a gray color and increased line spacing.

2. **Class Selectors**

   - Targets elements with a specific class attribute, marked by a period (`.`) followed by the class name.
   - Allows us to apply styles to multiple elements that share the same class, making it versatile for groups of elements.

   Example:

```
.highlight {
  background-color: yellow;
  padding: 5px;
}
```

   This rule applies a yellow background and padding to any element with `class="highlight"`.

3. **ID Selectors**

   - Targets a single, unique element with a specified ID, prefixed by a hashtag (`#`).
   - Useful for styling individual elements that require distinct styling, such as a main header or specific section.

   Example:

```
#main-header {
  font-weight: bold;
  font-size: 2em;
}
```

   Only the element with `id="main-header"` will display bold text in a larger font size.

4. **Attribute Selectors**

   - Targets elements based on their attributes (e.g., type, name, href).
   - Commonly used for form elements and links to apply styles based on specific attributes.

   Example:

```
```

```
input[type="text"] {
  border: 1px solid gray;
  padding: 4px;
}
```

This rule styles only `<input>` elements where `type="text"`, adding a border and padding for a cleaner appearance.

The Box Model – Margins, Padding, Borders

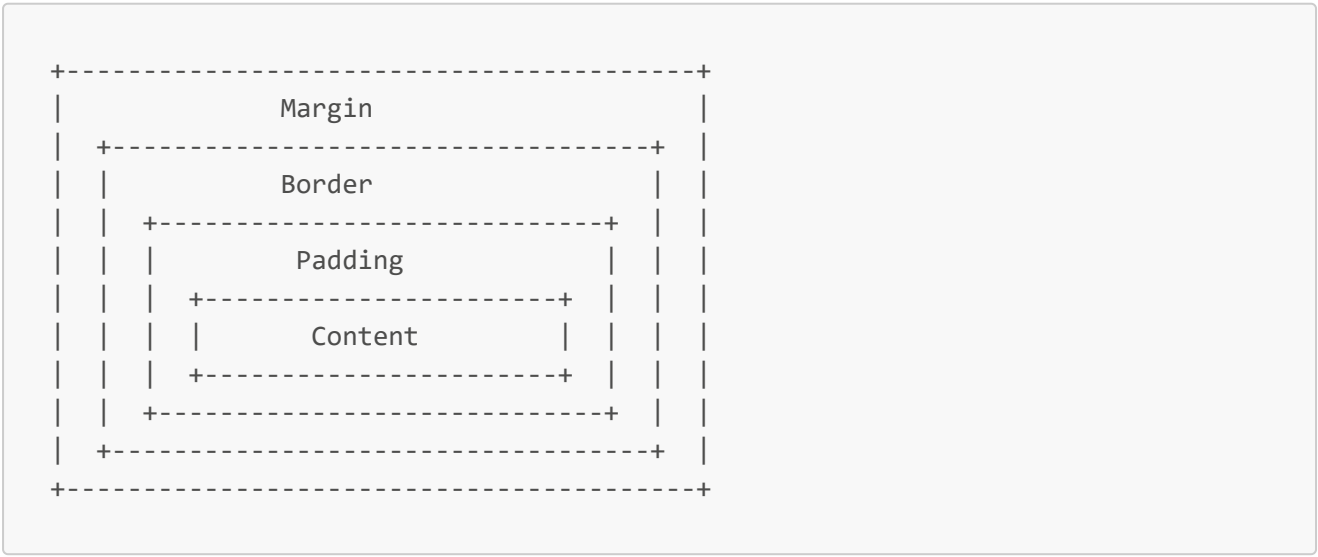
The **CSS Box Model** is a fundamental concept for controlling layout and spacing in web design. Each HTML element is considered a "box," and understanding this model allows us to manipulate the structure and spacing of elements precisely.

1. Overview of the Box Model

The box model views each HTML element as a rectangular box with four main areas surrounding the content:

1. **Content**: The actual content within the element, such as text or images.
2. **Padding**: The space between the content and the border. It creates an internal buffer around the content.
3. **Border**: A line surrounding the padding (and content), acting as a frame around the element.
4. **Margin**: The outermost space that creates distance between the element and neighboring elements.

Visualizing the box model:



2. Components of the Box Model

Content Area

- This is where the element's content (like text or images) appears.

- The content area is sized using properties like `width` and `height`.

  Example:

```
.box {
  width: 200px;
  height: 100px;
}
```

Padding

- Padding creates space inside the box, between the content and the border.
- The padding can be uniform or set individually for each side: `padding-top`, `padding-right`, `padding-bottom`, and `padding-left`.

  Example:

```
.box {
  padding: 10px; /* Adds 10px padding on all sides */
  padding-left: 15px; /* Adds extra padding on the left side */
}
```

Border

- The border surrounds both the padding and content.
- Borders have customizable thickness, style, and color, defined with `border-width`, `border-style`, and `border-color`.

  Example:

```
.box {
  border: 2px solid black; /* 2px thick solid black border */
}
```

- **Border Shorthand**: Using `border`, you can combine width, style, and color in a single line.

Margin

- Margin creates space outside the element, separating it from other elements.
- Like padding, margin can be set for all sides or individually: `margin-top`, `margin-right`, `margin-bottom`, and `margin-left`.

Example:

```
.box {
  margin: 20px; /* Adds 20px margin on all sides */
  margin-top: 10px; /* Sets a different margin for the top */
}
```

- **Margin Collapsing**: In CSS, vertical margins between elements can sometimes "collapse" or merge. This behavior reduces the total space between two elements if they both have top and bottom margins. This is unique to vertical margins and doesn't affect padding or horizontal margins.

3. Practical Exercises to Understand the Box Model

Exercise 1: Experiment with Padding and Border

- Create a box with text content and style it with different padding values to observe how padding changes the spacing inside the element.
- Add a border around the box and experiment with different thicknesses and colors.

Example:

```
.content-box {
  width: 150px;
  height: 80px;
  padding: 10px;
  border: 3px solid blue;
}
```

Exercise 2: Using Margins to Control Spacing Between Elements

- Place multiple elements (e.g., `<div>` boxes) on the page and adjust their margins to control the space between them.
- Try adding margins on different sides to see how they affect positioning relative to neighboring elements.

Exercise 3: Build a Simple Layout Using the Box Model

- Create a container with nested elements and use the box model to adjust spacing.
    - Add padding inside the container for internal spacing.
    - Use margins between the nested elements to separate them.
    - Add borders to visually outline each element.

      Example:

```css
.container {
  padding: 20px;
  border: 2px solid gray;
}

.nested-box {
  margin: 10px;
  padding: 15px;
  border: 1px solid black;
}
```

By the end of this section, you should understand how to use the box model to control spacing and layout, making your webpage structure more organized and visually appealing. This knowledge is essential for building complex layouts and achieving precise control over your webpage design.

Colors, Backgrounds, Borders, and Text Styling

This section focuses on applying colors, backgrounds, borders, and various text styling techniques to enhance the visual appeal of a webpage.

1. Text Styling: Making Text Polished and Readable

CSS offers a variety of properties to control how text looks on a webpage. These properties allow you to create visually appealing, readable content that matches the design goals of your website.

Key Text Styling Properties

1. **Color**
    - Sets the color of the text.
    - Can be defined using color names, HEX codes, RGB values, or HSL values.

    Example:

```css
p {
  color: #333333; /* Dark gray color */
}
```

2. **Font Size**
    - Controls the size of the text.
    - Typically set in pixels (`px`), ems (`em`), or rems (`rem`).

    Example:

```
h1 {
  font-size: 24px;
}
```

3. **Font Family**
   - Specifies the font to use for the text. You can choose from standard fonts or use custom fonts with `@font-face` or font libraries.
   - Fonts are usually listed in a fallback order.

   Example:

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

4. **Font Weight**
   - Adjusts the thickness of the text. Common values include `normal`, `bold`, `lighter`, and numerical values (100–900).

   Example:

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

5. **Text Alignment**
   - Sets the horizontal alignment of text within its container.
   - Options include `left`, `right`, `center`, and `justify`.

   Example:

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

6. **Line Height**
   - Controls the vertical spacing between lines of text, making text more readable and balanced.
   - Can be set as a unitless number or as a value in `px`, `%`, or `em`.

   Example:

```css
p {
  line-height: 1.5; /* Adds 50% more space between lines */
}
```

2. Backgrounds and Borders: Enhancing Elements with Color, Images, and Shape

Adding backgrounds and borders allows you to emphasize elements, create visual hierarchy, and improve the overall look and layout of the webpage.

Background Properties

1. **Background Color**
   - Sets a background color for an element, which can highlight sections or add depth to layouts.
   - Accepts color names, HEX codes, RGB values, or HSL values.

   Example:

```css
.highlight-section {
  background-color: #f0f8ff; /* Light blue background */
}
```

2. **Background Image**
   - Applies an image as the background of an element.
   - Can be combined with properties like `background-repeat` and `background-size` for control over how the image displays.

   Example:

```css
.banner {
  background-image: url("banner.jpg");
  background-size: cover; /* Image covers entire element */
  background-repeat: no-repeat;
}
```

Border Properties

1. **Border Width, Style, and Color**
   - `border-width`: Defines the thickness of the border.
   - `border-style`: Sets the style (e.g., `solid`, `dotted`, `dashed`, `double`).
   - `border-color`: Specifies the color of the border.

   Example:

```
.box {
  border-width: 2px;
  border-style: solid;
  border-color: #333;
}
```

2. **Border Radius**

   - Adds rounded corners to an element by setting the radius of each corner.
   - The higher the `border-radius` value, the more rounded the corners become. Setting it to `50%` creates a circle if applied to a square element.

   Example:

```
.profile-pic {
  border: 3px solid #ccc;
  border-radius: 50%; /* Makes a circular border */
}
```

3. Practical Exercises to Reinforce Text Styling, Backgrounds, and Borders

Exercise 1: Customize Text Appearance

- Create paragraphs and headings on the page and apply different text styling properties.
  - Experiment with color, font-size, font-family, and text alignment to see how each property affects readability and design.

Exercise 2: Apply Background Colors and Images

- Add background colors to various sections on the page to separate content areas visually.
- Use a background image for a specific section and control its position and size.

Exercise 3: Experiment with Borders and Border Radius

- Create several boxes with different border styles, colors, and widths.
- Add `border-radius` to see how rounded corners can change the look and feel of elements.

Summary

In this section, we learned how to make content more visually engaging with text styling, backgrounds, and borders. By combining these CSS properties, you can create a polished, cohesive look that enhances the readability and aesthetics of your webpage.

Using Git for Project Version Control

Version control is essential for any coding project, and **Git** is a widely-used system that allows developers to track and manage changes to their code. This session focuses on using Git to save and organize CSS styling changes. By the end of this section, you'll be comfortable committing changes, writing meaningful messages, and pushing updates to a remote repository like GitHub.

1. Overview of Git for Version Control

Git is a distributed version control system that helps developers

- Track changes in their code over time.
- Revert to previous versions if needed.
- Collaborate effectively by merging different contributions into a project.

In this session, we'll focus on:

- **Committing changes**: Saving specific modifications with a message describing what was changed.
- **Pushing changes**: Uploading saved changes to a remote repository, such as GitHub, to back up and share work.

2. Setting Up and Initializing a Git Repository

Before using Git for your project, you'll need to initialize a Git repository within your project folder. This turns the folder into a Git-tracked directory.

Step-by-Step Guide

1. **Navigate to the Project Directory**:
    - Use the terminal to move to your project folder.
    - Example command (for Mac or Linux):

```
cd /path/to/your/project
```

2. **Initialize Git**:
    - Run the following command to turn your project folder into a Git repository:

```
git init
```

    - This creates a hidden `.git` folder that Git will use to track your changes.

3. **Check the Repository Status**:
    - You can check the current status of your files with:

```
```

```
git status
```

- This command shows which files have been modified and which files are ready to be staged for a commit.

3. Committing Changes

A **commit** in Git is a snapshot of the project at a particular point in time. Each commit has an associated message to describe what was changed, making it easier to track the history of the project.

Steps to Commit Changes

1. **Stage the Files**:
   - Staging prepares the files you want to commit. You can stage all modified files with:

     ```
     git add .
     ```

   - Alternatively, you can add specific files by listing them individually:

     ```
     git add styles.css
     ```

2. **Write a Commit Message**:
   - Commit your staged changes with a meaningful message that describes what you've done.
   - Example command:

     ```
     git commit -m "Updated background colors and font styles in
     styles.css"
     ```

Tips for Writing Good Commit Messages

- Keep messages short and descriptive.
- Use a verb to indicate what was done (e.g., "Added," "Fixed," "Updated").
- Focus on **why** the change was made, rather than just what was changed.

4. Pushing Changes to GitHub

After committing your changes locally, you'll want to **push** them to a remote repository like GitHub. Pushing backs up your work and allows others to see your updates if they're collaborating on the project.

Steps to Push Changes

1. **Connect to a Remote Repository**:
    - If you haven't linked your project to a GitHub repository yet, you can add the remote link with:

```
git remote add origin https://github.com/yourusername/your-repo-name.git
```

2. **Push Your Changes**:
    - Use the `git push` command to upload your commits to GitHub. The `-u origin main` flag tells Git to push to the `main` branch on the `origin` remote.
    - Example command:

```
git push -u origin main
```

    - For subsequent pushes, you can simply use `git push` if you're working on the same branch.

5. Practical Exercises

Exercise 1: Commit Your CSS Changes

- Modify your CSS file, such as changing background colors or adjusting font sizes.
- Stage and commit these changes with a clear, descriptive message.
- Practice committing multiple times with different changes to understand the incremental nature of version control.

Exercise 2: Push Your Project to GitHub

- Set up a new repository on GitHub for your project.
- Link your local Git repository to the GitHub repository and push your commits.
- Check GitHub to confirm that your updates have been uploaded.

Exercise 3: Track Your Version History

- Use the `git log` command to view a list of previous commits.
- Review the messages and timestamps to see how your project evolved over time.

Git for Project Version Control Summary

Using Git for version control in your CSS project will allow you to track changes and collaborate effectively. By regularly committing and pushing updates, you'll ensure that every stage of your project's styling is documented and backed up. This practice is essential for maintaining organized and efficient workflows in software development.