CSS 101

CSS, an acronym for **Cascading Style Sheets**, is the language for styling web pages. It serves to define the visual presentation of web content. In the structure of web development, there are three distinct layers: content, presentation, and behavior.

HTML forms the content layer, setting up the structure and elements of a webpage like headings, paragraphs, lists, links, and images. JavaScript typically manages the behavior layer, enabling dynamic interactions in real time.

The role of CSS is pivotal in shaping the presentation layer, dictating how the HTML components look, covering aspects like color schemes, typography, backgrounds, and layouts. It's instrumental in ensuring that web content is responsive to different devices and screen sizes.

As we progress, we'll explore how to maintain a clear separation between these layers: HTML for content, CSS for design, and JavaScript for functionality. Before the advent of CSS, styling was directly embedded within HTML. This approach is now outdated, as it intertwines content structure with design.

There are compelling reasons to separate structure from style. Writing semantic HTML means using the most appropriate tags for your content's purpose rather than its appearance, which enhances meaning, accessibility, and search engine optimization. HTML was never meant for styling; hence, mixing style with structure leads to an inefficient workflow.

CSS streamlines website development and upkeep. Styles set in one CSS file are applied universally across multiple pages, allowing for site-wide design changes through a single file update.

CSS is ever-evolving, embracing advanced visual capabilities like rounded edges, shadows, gradients, animations, as well as responsive design tools such as multi-column layouts, grids, and Flexbox. As technology advances, CSS adapts, offering new features to tackle emerging web design challenges.

Adding CSS to HTML: Internal Styles

CSS can be applied to web pages in three ways:

1. through inline styles,
2. internal styles, and
3. external style sheets.

Here's a summary and evaluation of each method:

Inline Styles: Inline styles are written directly within an HTML tag using the **style** attribute. While this method allows for quick styling of individual elements, it's not recommended for comprehensive website development. Inline styles mix content with presentation, leading to scattered CSS that's hard to maintain. They also take precedence over other styles due to their position in the CSS cascade, making them difficult to override.

Internal (Embedded) Styles: Internal styles are placed within the **<style>** tag in the HTML document's head section. This method is suitable for styles that are unique to a single page. However, it still doesn't provide the separation of content and presentation that's ideal for larger websites.

External Style Sheets: External style sheets are the most efficient and widely accepted method. CSS rules are stored in separate **.css** files, which are linked to the HTML documents. This approach allows for a clean separation of content from style, making it easier to maintain and update the visual design across multiple pages.

When you start working with CSS, you'll see a default set of styles applied by the browser, known as user agent stylesheets, which make unstyled HTML look presentable. It's essential to understand how these default styles work and how they can be overridden by your custom styles. As you learn more about CSS, the concept of the cascade—how different styles interact and which ones take precedence—will become crucial.

Adding CSS to HTML: Internal Styles

The second method for incorporating CSS into a webpage is by using internal styles. These are placed within the **<style>** tag in the **<head>** section of the HTML document. For example, within the **<style>** tag, you might specify styles for paragraphs to increase their font size and make them bold, or set the font size for headings to make them stand out more.

However, if you had previously used inline styles for these elements, the internal styles wouldn't take effect because inline styles have higher specificity and would override the internal styles. Removing the inline styles would allow the internal styles to be applied, as seen when changing an **<h1>** tag's color from white to red after removing the inline style.

While internal styles can be convenient for very small websites or temporary changes, they are not ideal for larger projects. One of the main drawbacks is that if the styles are included in every HTML file, the browser has to reload these styles with each new page, leading to inefficiency and redundancy. This contradicts the DRY (Don't Repeat Yourself) principle that developers advocate for efficiency. Editing styles across multiple pages becomes cumbersome if you're not using an external stylesheet, which is a more efficient way to manage CSS for larger projects.

```
<head>
    <style>
        H1 {
           color:blue;
           font-size:14px;"
        }
    </style>
</head>
```

Adding CSS to HTML: External Stylesheets

From previous lessons, we've learned that while inline and internal styles are valid CSS methods, they're not ideal for larger projects due to the lack of separation between content and design, resulting in more complex development and maintenance.

The preferred approach is to use an external style sheet, which allows for site-wide design changes from a single file. This method involves removing any internal styles and creating a **.css** file, typically stored in a folder named 'css' for organization. You must then link this stylesheet to the HTML document using the **<link>** element within the **<head>** section, ensuring you set the **rel** attribute to 'stylesheet' and the **href** attribute to the path of the CSS file.

Using an external style sheet offers several benefits. It centralizes styling decisions, making updates simpler and more consistent across multiple pages. Moreover, once a stylesheet is downloaded by a browser, it's cached, speeding up load times for subsequent pages and visits. It's best practice to link stylesheets in the head section for efficient loading, ensuring the HTML content is styled before being displayed to the user. While it's possible to link multiple stylesheets, it's advisable to limit them to reduce server requests and maintain website performance.

CSS Selectors

This section introduces the basics of writing CSS rules and understanding their importance in web development. Here's a summary:

A CSS rule is composed of two main parts: the selector and the declaration block. The selector targets the HTML element to be styled, much like a street address identifies a house. The declaration block, enclosed in curly brackets, contains styling instructions for the selected elements.

Types of CSS Selectors:

- **Universal Selector**: Selects all elements on a webpage. It’s symbolized by the asterisk (*) and is useful for applying a style universally.
- **Type Selector**: Targets all instances of a specific HTML element. For example, **p** will select all paragraph elements.

- **Class Selector**: Selects elements with a specified class attribute, using a period (.) followed by the class name.
- **ID Selector**: Targets a single element with a specific ID attribute, using a hash (#) followed by the ID name.
- **Descendant Selector**: Selects elements that are descendants of a specified element. For example, **#intro img** targets **img** elements within the element with ID **intro**.
- **Grouping Selectors**: Allows for applying the same styles to multiple HTML elements, separated by commas.

Understanding the cascade in CSS is crucial. The cascading order determines which styles are applied when there are conflicting rules. It can be summarized as follows:

- Styles declared later in the CSS file take precedence over earlier ones.
- Class selectors have a higher priority than type selectors and order in the file.
- ID selectors have the highest priority and will override both class and type selectors.

The specificity of selectors (ID being the most specific, followed by classes, and then type selectors) plays a key role in determining which styles are applied to an element when multiple rules conflict.

Having practiced with selectors and understood the concept of specificity, the next stage is to learn about writing the actual styling code within the declaration blocks of these selectors. This introduction to CSS rules and selectors lays the foundation for creating visually appealing and well-structured web pages, with an emphasis on the importance of the cascade and specificity in CSS.

See: css-1 folder

In CSS, a declaration block contains declarations, each consisting of a CSS property name and its value, separated by a colon and ending with a semicolon. This block is where we specify how to style the selected HTML elements.

Let's explore three fundamental CSS properties:

- **Color**: This property sets the text color of an HTML element. For example, **color: red;** sets the text color to red.
- **Background Color**: This property controls the background color of an element. For instance, **background-color: green;** sets the background color to green.
- **Border**: This property defines an element's border and requires three values: color, width, and style (e.g., **border: black 3px solid;**).

For example, using the universal selector, we set all elements to have white text, a black background, and a white border. This dramatically changes the webpage's appearance.

However, It's rare to apply the same styles to all elements. Instead, we use more specific selectors like the body or class selectors to apply styles more appropriately.

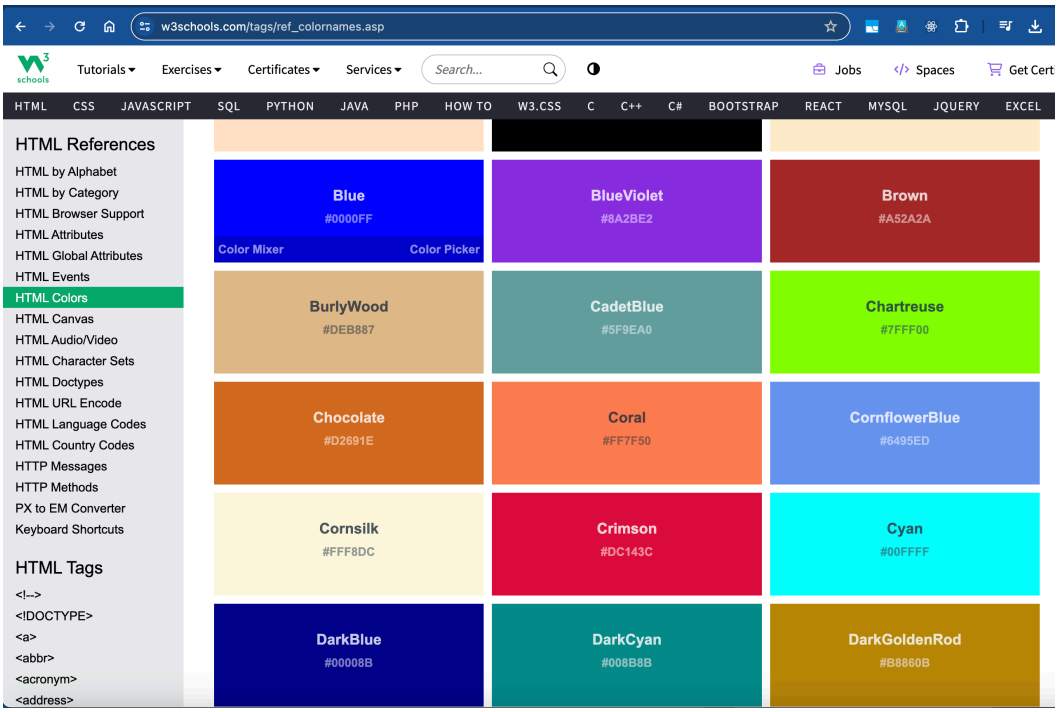
This segment effectively illustrates how CSS properties can be used to enhance the visual aesthetics of a website, showcasing the fundamental principles of web design and the importance of CSS in creating appealing and effective web pages.

See: css-2 folder

CSS Colors

So far, we have learned how to apply basic HTML color names to elements in our web design. These color names, like white, black, brown, and rosy brown, offer a simple and readable way to define colors. However, for more specific color choices, we turn to hexadecimal (hex) colors.

Hexadecimal colors provide a precise way to represent colors using the format **#RRGGBB**, where **RR** is the red component, **GG** is green, and **BB** is blue. Each pair can range from **00** to **FF**, allowing for an extensive palette of colors. For example, **#FF0000** represents pure red.

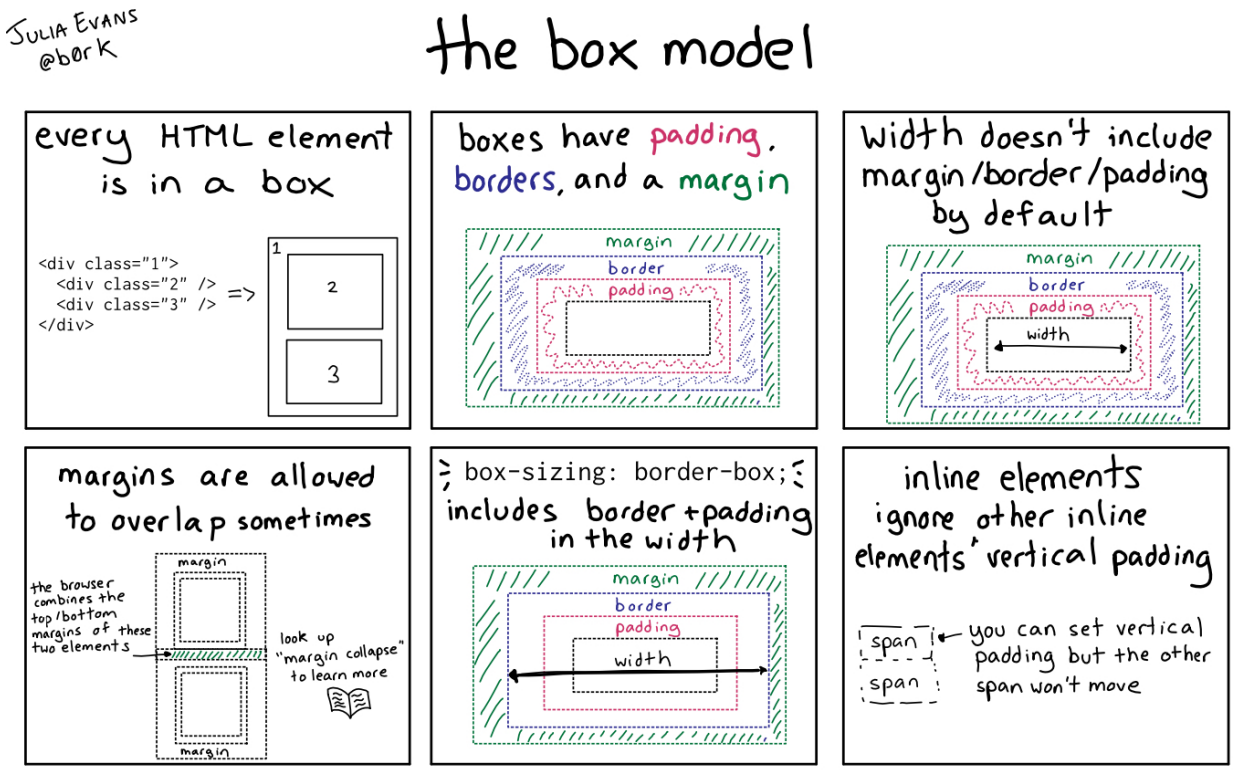


Applying Hex Colors in CSS: We replace the simple HTML color names in our CSS with hex colors for more accuracy. For instance, changing the body's background color from rosy brown to a specific light brown (**#E8DBD3**) found in the project's color.txt file.

This approach to using hex colors enhances the precision and customization of web designs, allowing for a more professional and tailored appearance of web pages.

CSS Box Model

CSS box model is a fundamental concept in web design, to style HTML elements effectively. The CSS box model conceptualizes HTML elements as boxes, consisting of the content area, padding, border, and margin. This model is crucial for defining the layout and dimensions of elements on a webpage. It defines the following properties:



Width and Height Properties: HTML elements have width and height properties, determining the size of the content area. Adjusting these properties can significantly alter the space an element occupies. For instance, setting the width of an image to 200 pixels makes it narrower.

Max-width and Max-height: These properties set maximum dimensions for elements, ensuring they don't exceed a certain size, which is particularly useful for responsive design.

Content Area: The area where the element's content, such as text or images, is displayed.

Border: A visible boundary around the element, customizable in style, color, and width. The

border-radius property can be used to create rounded corners.

Padding: The space between the content and the element's border. It can be set uniformly or specified for individual sides (top, right, bottom, left).

Margin: The space outside the element, controlling the separation from other elements. Margins can be set uniformly or for specific sides, and the **auto** value is used to horizontally center elements.

This comprehensive overview of the CSS box model and its properties provides a strong foundation for creating well-structured and visually appealing web layouts.

See: css-3 folder

Text and Fonts

Here's a summary of the key properties related to text and fonts:
Font-Related CSS Properties:

- **Font Family**: Specifies the typeface or font for text. For instance, **font-family: sans-serif;** applies a modern sans-serif font to the text.
- **Font Size**: Controls the size of the text, crucial for readability and visual hierarchy. Example: **font-size: 72px;**.
- **Font Weight**: Determines the thickness or weight of the text, creating contrast and emphasis. Example: **font-weight: bold;**.

Font Stacks: A font stack is a list of fonts in a web browser's order of preference. It ensures that if a specific font isn't available on the user's device, an alternative is used, maintaining consistency across different platforms.

Using Google Fonts: Google Fonts provides a library of free-to-use fonts.

Applying the Font Stack: We added the Google Font to our CSS file and specified it in the **Line-35** selector to apply it to the corresponding line.

Text Styling Properties:

- **Text-align**: Aligns text within its container, similar to text alignment in word processors. Example: **text-align: center;**.
- **Line-height**: Controls the vertical space between lines of text, enhancing readability. Example: **line-height: 2;**.

This tutorial emphasizes the importance of typography in web design and provides practical skills for enhancing the visual appeal and readability of web content using CSS.

See: css-4 folder

CSS Positioning

Let's explore the CSS **display** property and how it affects the layout and positioning of elements on a web page. Here's a summary of the key points:

The Display Property: Determines how an element is displayed on a web page. The key values of this property are:

- **Block**: Elements start on a new line and take up the full width available. Examples include **body**, **div**, and **p**.
- **Inline**: Elements only take up as much width as their content requires. Top and bottom margins and paddings do not affect inline elements. Examples are **a** and **img** tags.
- **Inline-block**: Combines features of both block and inline elements, allowing for margin and padding customization without starting on a new line.

Centering an Image: To center an image that is an inline element by default, we first change its display property to **block**. Then, we apply **margin: 0 auto;** to center it.

Styling Anchor Tags: Anchor tags are inline elements by default. We styled a specific anchor tag by adding an ID (**back-to-top**) and using the ID selector in CSS. We set its display to **block** and aligned the text to the right.

Customizing Navigation Link: We transformed a navigation link into a block-level element for styling purposes. However, we encountered an issue as block elements take up the full width of their container. We resolved this by changing the display property to **inline-block**, allowing for padding and background color application while maintaining the content's width.

Applying Advanced Styles: We applied various styles to the navigation link, including background color, padding, border-radius for rounded corners, and font adjustments (color and weight).

CSS Images

Let's address the issue of the image size that have larger or smaller size than what we aim to have. Here's a summary of the key points that sets the properties of images:

Responsive Image Sizing:
To make the image responsive, we used the **max-width** property, setting it to **100%** of the image's container. This adjustment ensures the image shrinks to fit within its container when necessary but retains its original dimensions when possible.

See: https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_image

Using percentage units for image sizing is a part of responsive web development, which aims to make web content adapt to different screen sizes.

Adding a Background Image:
We can add a background image to the body element using the **background-image** property.

The URL notation can be used to specify the path to the image file.

Fixing the Background Image:
To prevent the background image from moving during scrolling, which can be visually distracting, we used the **background-attachment** property with the value **fixed**. This made the background stay stationary while scrolling.

See: https://www.w3schools.com/cssref/tryit.php?filename=trycss_background-attachment

Class Exercise:

You can find many useful examples at:

https://css-examples.wizardzines.com/#inline-block-looks

Check as many examples as you like. Make some changes to the code, until you understand the concepts. Ask question if you do not understand a part of find a part difficult.

You can open all codes in codepen and see and practice the code interactively live.