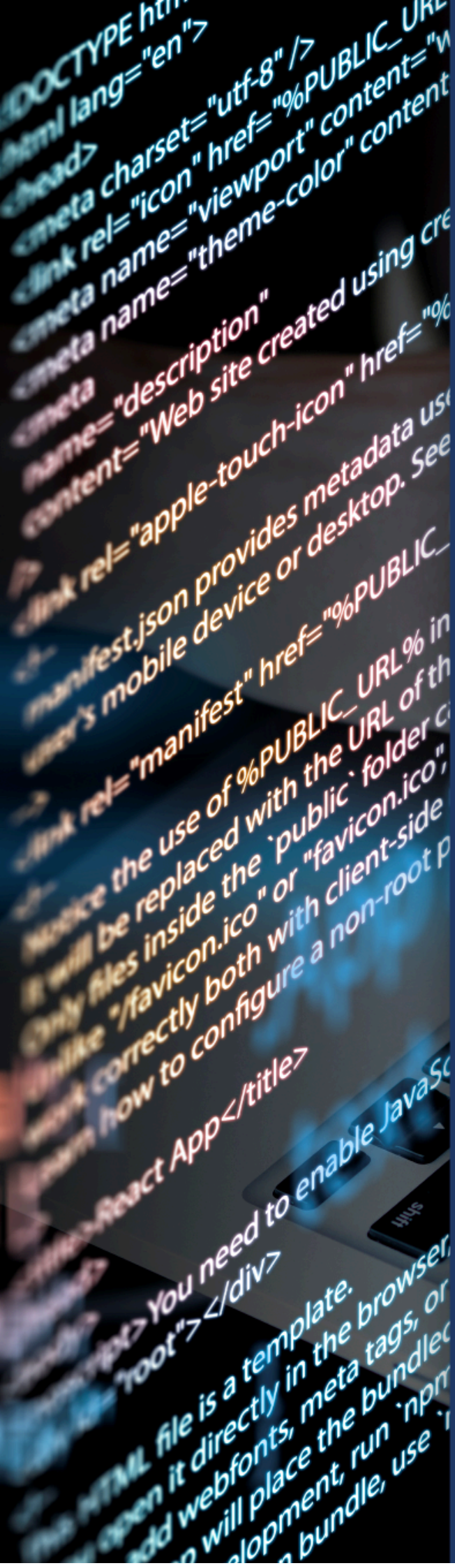


FULL STACK
FRONTEND

CSS

SHAI
Share Artificial Intelligence

1st version, 2025 @SHAI

Full Stack Track

CSS

Content Team

- **Subject Matter Expert:** Roaa Abu-Shaqrah
- **Instructional Design:** Dyala Alrefai , Jumana Hamad, Maroom Alrefai

Note: Some parts of the content were generated using AI tools and have undergone thorough review and verification by the expert SHAI team to ensure accuracy.

This self-evaluation table is designed to help learners assess their knowledge, skills, and performance in CSS both before and after completing the unit. It includes a list of key CSS competencies, covering foundational concepts, practical applications, and best practices. Learners will rate their confidence level for each skill or knowledge area on a scale of 1 to 5, where:

- **1** = Not Confident
- **5** = Highly Confident

Before beginning the unit, complete the Pre-Self Evaluation column. After finishing the unit, revisit the table to reflect on your learning progress. This process will help you assess your development and identify any concepts or skills that may need further improvement.

Skills/ Knowledge	Pre-Self Evaluation	Post-Self Evaluation
Understanding the role of CSS in web development and its relationship with HTML		
Explaining how browsers interpret CSS to render styled elements		
Defining key CSS concepts such as selectors, properties, and values		
Applying basic CSS rules to style HTML elements effectively		
Differentiating between universal, element, class, and ID selectors		
Writing CSS rules to style text (e.g., font size, color, boldness)		
Specifying colors using RGB and HEX codes		
Styling text with padding, margin, and borders for better layout		
Understanding and applying the CSS box model		
Customizing ordered and unordered list styles with CSS		
Styling tables with borders, padding, and alignment		
Designing visually appealing and responsive forms using CSS		
Positioning elements using static, absolute, relative, fixed, and sticky properties		
Using CSS Grid for two-dimensional layouts		
Applying Flexbox for one-dimensional layouts		
Creating responsive designs with media queries		
Using the BEM naming convention for scalable and maintainable CSS		

Learning objectives	Getting started
This chapter will guide you through the fundamental concepts and techniques essential for mastering CSS, enabling you to design visually engaging, responsive, and well-structured web pages with effective styling and layout practices. By the end of the chapter, you will be able to: -Learn the basic structure of CSS (selectors, properties, and values) and understand different types of selectors (element, class, ID, attribute, pseudo-classes, and pseudo-elements). -Understand the **box model** and learn how to create complex layouts using **Flexbox** and **CSS Grid** for responsive design. -Master styling properties for text (font, color, size, etc.) and elements (background, borders, margins, padding), as well as advanced styling techniques like shadows, gradients, and animations. -Learn how to use **media queries** to create designs that adapt to different screen sizes and devices, utilizing relative units like percentages, em, rem, and viewport units. -Understand the different positioning schemes (static, relative, absolute, fixed, sticky) and how to align elements within containers using **Flexbox** and **Grid**. -Write maintainable, optimized, and clean CSS by using techniques like **BEM** (Block Element Modifier), managing specificity, and optimizing performance with minification and compression	I have finished the newspaper project in HTML but in black and white and we also talked about the Wikipedia encyclopedia. How can we build a more attractive site with colors and moving images to attract users? In this book we will start learning to design attractive sites. Let's go.  *Figure #: Wikipedia man looks to your project and think "Should I email him".*

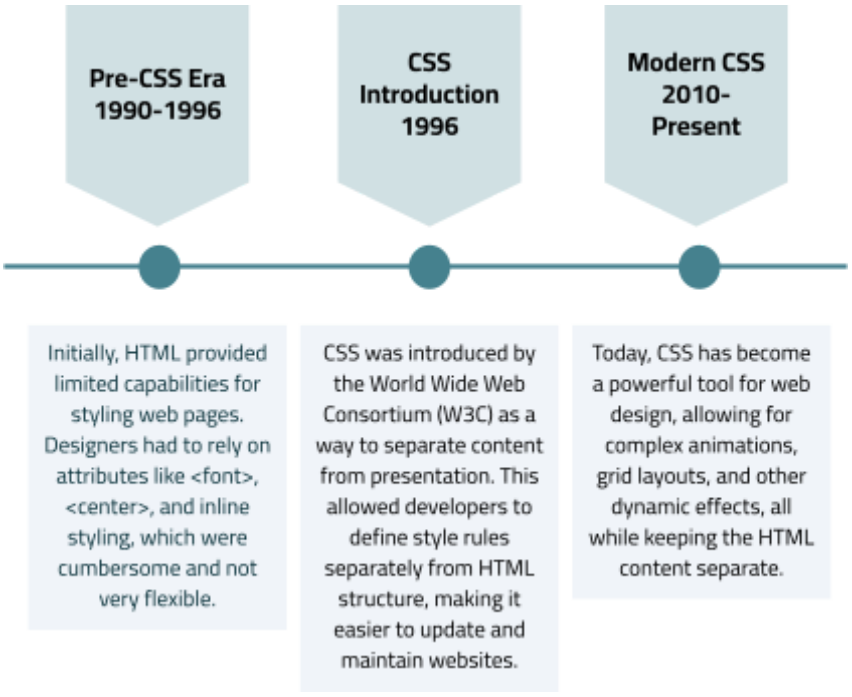
1. Introduction

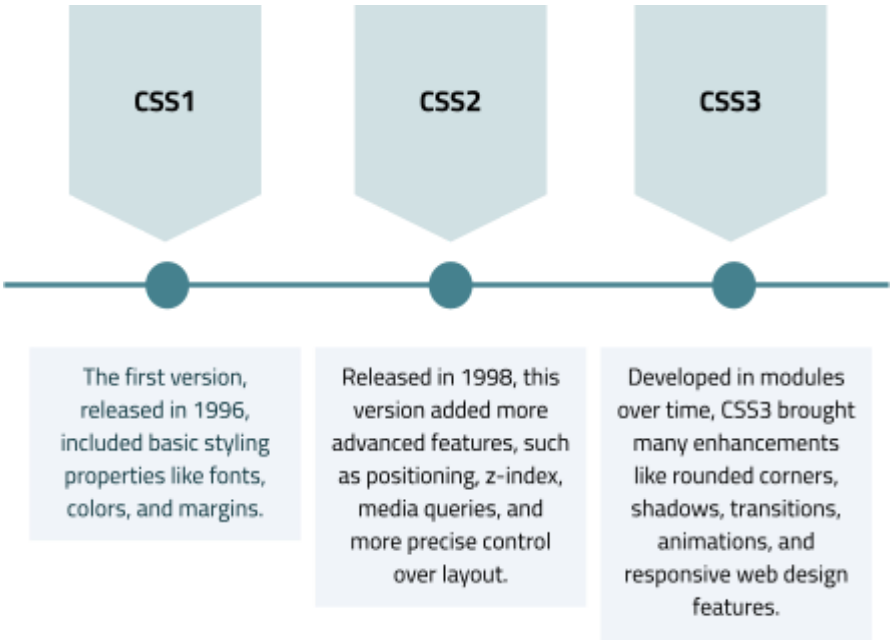
Imagine yourself as a designer walking into a blank room. This room has no color, no furniture, and no decoration. It's just empty walls and floors, waiting for your creative touch. You start by deciding on the theme. Will it be modern, vintage, or minimalist? You then pick out the paint colors for the walls, decide where to place the furniture, and arrange everything to create a cohesive look. You can even add some fancy lighting to set the mood.

figure#: Man realized life doesn't have Ctrl+Z, so he sat down and thought harder than ever.

In the world of web design, this blank room is like an HTML page—structured, but bare. Now, you are the CSS designer. You have the power to style this empty page. You choose colors, fonts, and layouts to make everything look polished and attractive. Just like you'd place a chair here or a lamp there in the room, you position elements on the web page with precision using CSS.

1.1 History





1.2 What does CSS

CSS (**C**ascading **S**tyle **S**heets) is a style sheet language used to describe the presentation of a document written in HTML or XML. It controls the layout, design, colors, fonts, and overall appearance of web pages. While HTML structures the content of a web page, CSS defines how that content should look.

1.3 HOW CSS Work

CSS (Cascading Style Sheets) is a fundamental technology used to control the look and feel of web pages. It allows you to apply styles to HTML elements, such as colors, fonts, layout, and more. Understanding how CSS works can help you effectively design and manage the presentation of your web pages.

2. Basic Concepts of CSS

CSS concepts refer to the foundational principles and techniques used to style and layout web pages, including how to control the visual presentation of HTML elements, manage colors, fonts, spacing, and responsiveness, creating an engaging and user-friendly interface.

Selectors	Properties and Values	Cascade	Inheritance	Specificity
CSS targets HTML elements using selectors.	CSS rules define how elements should look by setting properties with specific values.	CSS follows a "cascading" mechanism to resolve conflicts when multiple styles apply to the same element.	Some properties are inherited from parent elements by their child elements.	CSS determines which styles to apply based on specificity rules when multiple conflicting rules exist.

Write down the Properties and Selector from your study of HTML before we start explaining them.

_____
_____
_____

3. Types of CSS Selectors

In CSS, a **selector** is a pattern used to select and target HTML elements that you want to style.

The selector determines which HTML elements the styles will apply to.

1. Universal Selector (*)

The universal selector selects all elements in a document.

Watch Out
Using too many universal styles can make your CSS hard to debug and maintain.

```
{
 color: red;
}
```

2. Type (Element) Selector

This targets all elements of a specific type (like div, p, h1, etc.).

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

3. Class Selector (.)

The class selector targets elements with a specific class attribute. It is preceded by a dot (.).

```
.example {
 background-color: yellow;
  }
```

4. ID Selector (#)

The ID selector targets a single element with a specific ID attribute. It is preceded by a hash (#).

```
#header {
    font-weight: bold;
  }
```

5. Attribute Selector

This targets elements based on the presence or value of an attribute.

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

6. Descendant Selector (Space)

This selects elements that are nested inside a specific element.

```
div p {
   color: green;
  }
```

7. Child Selector (>)

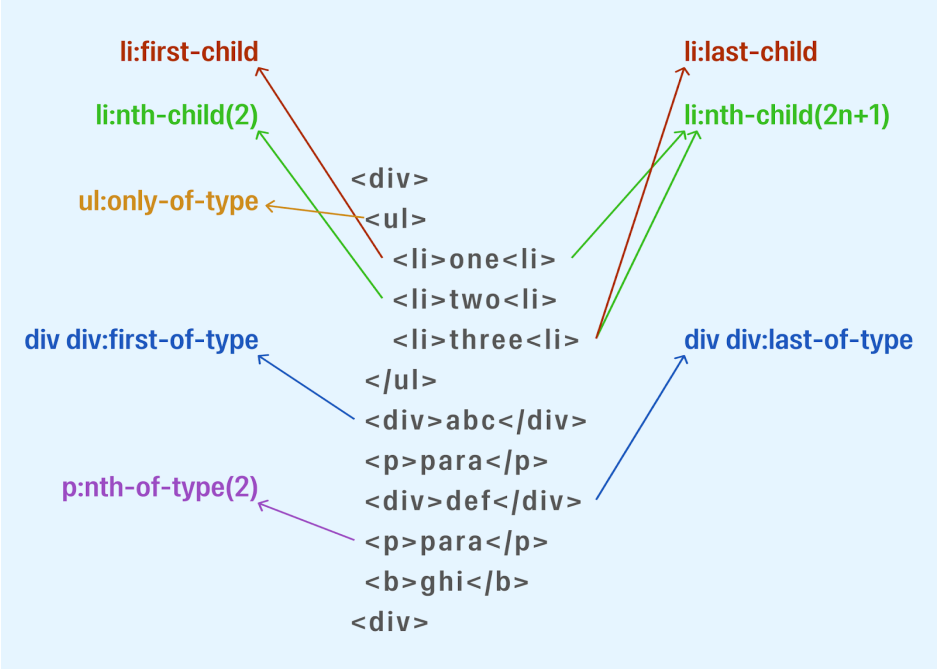
This targets elements that are a direct child of another element.

```
div > p {
   margin-top: 10px;
  }
```

8. Adjacent Sibling Selector (+)

This selects an element that is immediately preceded by a specific element.

```
h1 + p {
   font-size: 18px;
  }
```



9. General Sibling Selector (~)

This selects all elements that are siblings of a specific element.

```
h1 ~ p {
   color: blue;
  }
```

10. Pseudo-classes

Pseudo-classes target elements based on their state or position, such as :hover, :first-child, :nth-child(), etc.

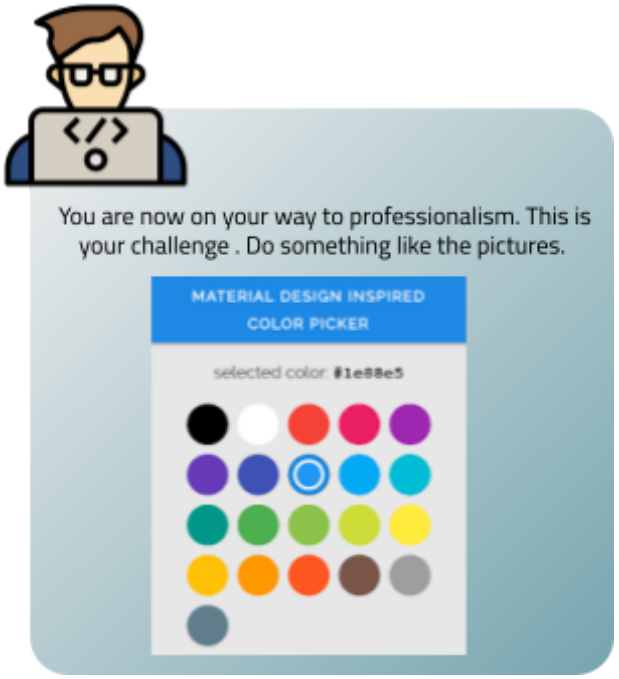
```
a:hover {
   text-decoration: underline;
  }
```

	Find out how to use the :nth-child pseudo-class to style alternating rows in a table.

11. Pseudo-elements

Pseudo-elements are used to style specific parts of an element, such as the first letter, first line, or before/after content.

```
p::first-letter {
    font-size: 2em;
  }
```



4. Rules, Properties , and Values

CSS is a language used to describe the presentation of a document written in HTML or XML. It controls the layout, colors, fonts, and spacing of elements. CSS has several key concepts, including **rules**, **properties**, and **values**.

4.1 CSS Rules

A **CSS rule** consists of a selector and a declaration block. The rule defines how HTML elements should be styled.

Structure:

Syntax	selector { property: value; }
Selector	It targets an HTML element or a group of elements.
Declaration Block	It contains one or more property-value pairs, enclosed in curly braces {}.

Example:

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

In this case:

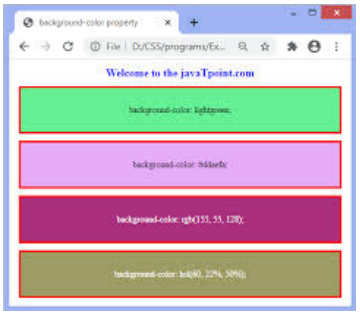
- p is the selector (it selects all <p> elements).
- color and font-size are properties.
- red and 16px are the corresponding values.

4.2 CSS Properties

Properties are the style features you want to modify. Each property controls a specific style aspect of an HTML element. Some properties affect the layout, while others impact the color, font, and visual effects.

Common CSS Properties:

- **Color Properties**
  - color: Sets the text color.
  - background-color: Sets the background color of an element.
  - border-color: Specifies the color of an element's border.

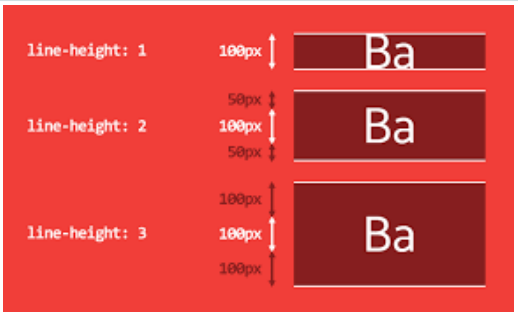
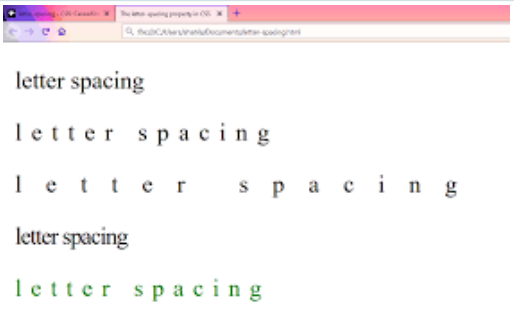


- **Font Properties**
  - font-family: Defines the typeface to be used.
  - font-size: Sets the size of the text.
  - font-weight: Specifies the thickness of the font (e.g., bold).

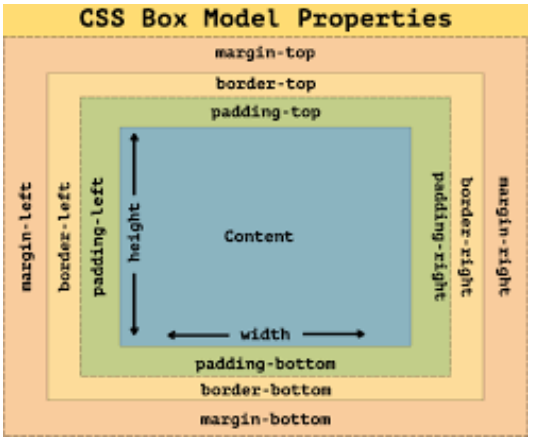
font	syntax
font color	font-color: Blue
font size	font-size: 48px;
font weight	font-weight: lighter;
font family	font-family: Noto Sans
font style	font_style: italic
font spacing	letter-spacing: 5px

- **Text Properties**

text-align: Aligns text within an element (left, right, center).	

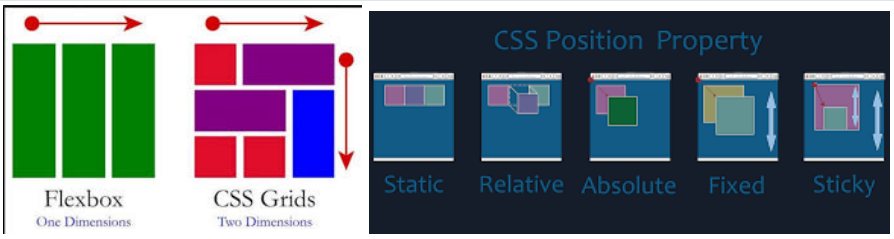
line-height: Adjusts the line spacing in text.	
letter-spacing: Controls the space between characters.	

- **Box Model Properties**



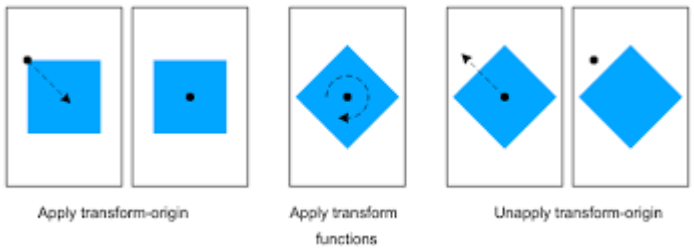
width	Defines the width of an element.
height	Sets the height of an element.
padding	Adds space inside an element, between the content and its border.
margin	Creates space outside the element, between it and other elements.
border	Sets the style, width, and color of the element's border.

- **Layout Properties**



display	Specifies the display style of an element (e.g., block, inline, flex, grid).
position	Defines how an element is positioned (e.g., static, relative, absolute, fixed).
flex	Used in flexible box layouts to adjust elements within a container.
grid	Creates space outside the element, between it and other elements.
border	Used in CSS Grid Layout for controlling rows and columns.

- **Transform and Animation Properties**



transform	Allows you to rotate, scale, skew, or translate an element.
transition	Defines the change of property values over time.
animation	Controls animations, like keyframes and duration.

Find out how to use CSS animations to create a "loading spinner." What properties and keyframes are required? Try implementing it in your project.

4.3 CSS Values

Values are the settings assigned to CSS properties. Each property requires a specific type of value, such as keywords, measurements, colors, or URLs.

Types of Values:

- **Keywords**: Words that specify certain property values.

- Examples: center, red, none, bold, auto.

- **Length values**: Used for dimensions such as width, height, padding, margin.
  - Units: px (pixels), em, rem, %, vh (viewport height), vw (viewport width), cm, etc.
  - Example: 10px, 2em, 50%.

Note
Use em or rem for font sizes to ensure scalability and accessibility.

How do **px**, **em**, **rem**, and **%** differ in CSS, and when should you use each one?

- **Color values**: Can be specified using:
  - Named colors: red, blue, green.
  - Hexadecimal values: #FF5733 (Hex for RGB colors).
  - RGB and RGBA: rgb(255, 0, 0) (RGB), rgba(255, 0, 0, 0.5) (RGBA, with transparency).
  - HSL: hsl(0, 100%, 50%) (Hue, Saturation, Lightness).
- **URL**: Used for linking resources, such as background images.
  - Example: url('image.jpg').

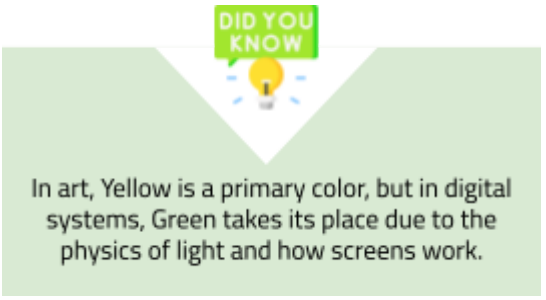
Note
When using CSS transitions, ensure you specify transition-duration; otherwise, the effect won't work.

- **Percentage**: Represents a percentage of a parent element or viewport.
  - Example: width: 50%.
- **Time values**: For animations, transitions, etc.
  - Examples: 1s, 500ms (seconds or milliseconds).
- **Other Values**:
  - **auto**: Automatically calculated value (e.g., for margins, width).
  - **inherit**: Makes an element inherit the style from its parent element.
  - **initial**: Resets a property to its default value.
  - **unset**: Resets a property to its inherited value if it's inheritable, or to its initial value if not.

What is the benefit of each auto , inherit ,initial , unset

5. How to specify colors

5.1 RGB (Red, Green, Blue)

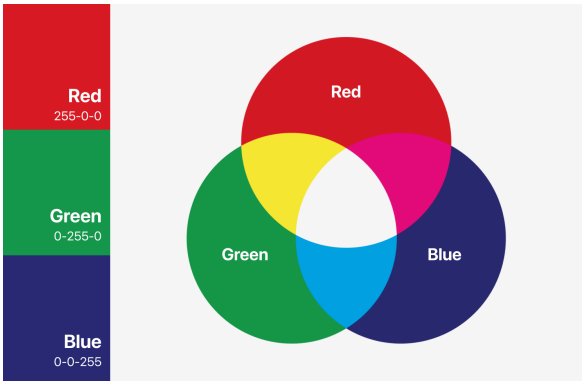


RGB is a color model used to represent colors on electronic displays such as computer monitors, TVs, and cameras. The model is based on the idea that different colors can be created by mixing three primary colors: **Red**, **Green**, and **Blue**. These colors are combined in varying intensities to produce a broad spectrum of colors.

Basic Concept

The RGB model works by mixing different intensities of the three primary colors:

- **Red** (R)
- **Green** (G)
- **Blue** (B)



Each of the colors is represented by a numerical value, usually ranging from **0** to **255**, which determines the intensity of that particular color. The higher the number, the more intense the color:

How It Works

- **0** means no light (i.e., black).
- **255** means the full intensity of that color.
- By combining different intensities of Red, Green, and Blue, you can create any color in the RGB spectrum.

RGB Color Representation

In **RGB**, a color is represented as a combination of three values (R, G, B). Each value can range from 0 to 255, so the total number of possible colors is 256 x 256 x 256 = **16,777,216** possible colors.

RGB Values Breakdown

- **R (Red)**: Controls the amount of red in the color. The value ranges from **0** (no red) to **255** (full red).
- **G (Green)**: Controls the amount of green in the color. The value ranges from **0** (no green) to **255** (full green).
- **B (Blue)**: Controls the amount of blue in the color. The value ranges from **0** (no blue) to **255** (full blue).

Common RGB Color Examples

- **RGB(255, 255, 255)**: White (full intensity of red, green, and blue).
- **RGB(0, 0, 0)**: Black (no intensity of any color).
- **RGB(255, 0, 0)**: Red (full red, no green or blue).
- **RGB(0, 255, 0)**: Green (full green, no red or blue).
- **RGB(0, 0, 255)**: Blue (full blue, no red or green).
- **RGB(255, 255, 0)**: Yellow (full red and green, no blue).
- **RGB(0, 255, 255)**: Cyan (full green and blue, no red).
- **RGB(255, 0, 255)**: Magenta (full red and blue, no green).

RGB in Web Design (CSS)

In CSS, RGB is used to define the color of elements. You can use the rgb() function to specify colors:

```css
h1 {
   color: rgb(255,0,0); /* Red */ }

button {
   background-color: rgb(0,255,0); /* Green */  }

div {
   border: 2px solid rgb(0,0,255); /* Blue */   }
```

5.2 Hexadecimal Color System (Hex)

The **Hexadecimal (Hex)** color system is a widely used way of representing colors in digital graphics and web design. It's often used as an alternative to RGB (Red, Green, Blue) for specifying colors in CSS and other design tools.HEX.

What is Hexadecimal?

The hexadecimal system is base-16, meaning it uses 16 distinct symbols. These symbols are:

- **0–9**: Represent the values 0 to 9.
- **A–F**: Represent the values 10 to 15.

A **hex color code** is a 6-character string representing colors. It's usually prefixed with a hash (#) symbol. The six characters are broken down into three pairs, each representing one of the primary colors: **Red**, **Green**, and **Blue**.

Each pair of characters is a **two-digit hexadecimal number** representing the intensity of that color component (R, G, B) on a scale from 0 to 255.

Structure of a Hex Color Code

#RRGGBB:

RR	Represents the Red component (in hexadecimal).
GG	Represents the Green component (in hexadecimal).
BB	Represents the Blue component (in hexadecimal).

Hexadecimal to RGB Conversion

The two characters in each pair are hexadecimal values (ranging from 00 to FF), which can be converted to a decimal (base-10) number ranging from 0 to 255. This value represents the intensity of the respective color.

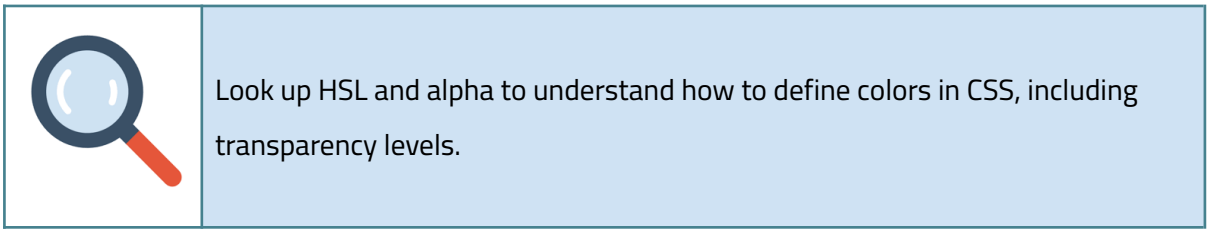
- **00** in hexadecimal equals **0** in decimal (no intensity).
- **FF** in hexadecimal equals **255** in decimal (maximum intensity).

Example: Hex to RGB Conversion

Let's break down the hex color #FF5733:

1. **Red**: The first pair FF represents Red.
    - FF in hexadecimal is 255 in decimal.
    - So, the Red component is 255.
2. **Green**: The second pair 57 represents Green.
    - 57 in hexadecimal is 87 in decimal.
    - So, the Green component is 87.
3. **Blue**: The third pair 33 represents Blue.
    - 33 in hexadecimal is 51 in decimal.
    - So, the Blue component is 51.

Therefore, the color #FF5733 in RGB is rgb(255, 87, 51).



6. Text Styling in CSS

In CSS, you can control various aspects of text, such as its size, weight (bold), style (italic), and spacing. These properties allow you to customize how text appears on your web page to improve readability and create a desired visual effect. Let's break down the difference .

6.1 Text Size (font-size)

The font-size property controls the size of the text. It can be defined in various units:

- **px**: Pixels, a fixed unit.
- **em**: Relative to the parent element's font size.
- **rem**: Relative to the root element (<html>), often used for consistent scaling.
- **%**: Percentage of the parent element's font size.
- **vw/vh**: Viewport width/height (can scale text according to screen size).

Example:

```
h1 {
    font-size: 36px; /* fixed size in pixels */
  }

p {
    font-size: 1.5em; /* Relative to the parent element */
  }

div {
    font-size: 2rem; /* Relative to the root element */
  }
```

6.2 Bold Text (font-weight)

The font-weight property is used to control the boldness of the text. Common values for this property include:

- **normal**: Default font weight.
- **bold**: Makes the text bold.
- **bolder**: Makes the text bolder than the parent.
- **lighter**: Makes the text lighter than the parent.
- **100 to 900**: Numeric values that allow fine control over the weight (100 being the lightest and 900 being the heaviest).

Example:

```
h1 {
    font-weight: bold; /* Bold text */
  }

p {
    font-weight: 700; /* Equivalent to 'bold' */
  }

strong {
    font-weight: 900; /* Very bold text */
  }
```

6.3 Italic Text (font-style)

The font-style property is used to make text italicized. Common values include:

- **normal**: Default, regular text style.
- **italic**: Makes the text italic.
- **oblique**: Similar to italic but may be less curved (often used for fonts that don't have a distinct italic style).

Example:

```
h1 {
    font-style: italic; /* Italicized text */
  }

p {
    font-style: oblique; /* Slightly slanted text */
  }
```

6.4 Spacing Between Lines (line-height)

The line-height property controls the amount of space between lines of text within a block. It is commonly used to improve the readability of paragraphs.

- **Normal**: The default line height is typically 1.2 to 1.5 times the font size.
- **Unitless value**: A number like 1.5 which is a multiplier of the font size.
- **px, em, rem**: Specific units to define the line height.

Example:

```
h1 {
    line-height: 1.8; /* 1.8 times the font size */
  }

p {
    line-height: 40px; /* Fixed line height */
  }
```

6.5 Spacing Between Words (word-spacing)

The word-spacing property controls the amount of space between words in a block of text. You can use it to increase or decrease the gap between words.

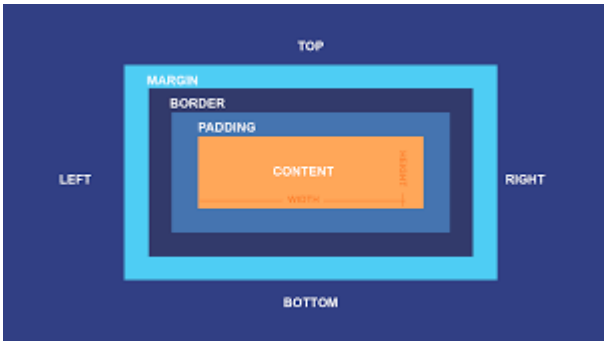
- **Normal**: The default word spacing.
- **Length values**: You can specify a length, such as px, em, or rem, to set the exact space between words.

Example:

```
p {
    word-spacing: 10px; /* Increases space between words */
  }

h2 {
    word-spacing: -2px; /* Decreases space between words */
  }
```

Find a way to leave space between letters.

7. Box model



The **CSS Box Model** is the fundamental concept that describes how elements are structured and spaced on a web page. Every element on a page is represented as a rectangular box, and the box model defines how its content, padding, border, and margin are arranged.

Here's how it works:

1. **Content**: The actual content of the element, such as text, images, or other media.
2. **Padding**: The space between the content and the element's border. It clears space around the content.
3. **Border**: A line that surrounds the padding (if specified) and the content.
4. **Margin**: The space outside the border, separating the element from other elements.
5. **Width and height**: By default, the width and height properties apply to the **content** area only. The padding, border, and margin are added around it.
6. **Total box size**: Total element size is calculated as:
    a. **Total width** = content width + left padding + right padding + left border + right border + left margin + right margin
    b. **Total height** = content height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

Note
Always reset margins and paddings with a CSS reset or normalize stylesheet to avoid inconsistent spacing across browsers.

Watch Out
Avoid setting fixed heights on containers holding dynamic content; it can cause overflow issues.

Total width would be ?

Content width: 200px **Padding**: 10px (all sides)

Border: 5px (all sides) **Margin**: 15px (all sides)

8. List Styling in CSS

Lists in HTML can be styled using the <ul>, <ol>, and <li> elements. CSS allows you to customize the appearance of lists, including the type of bullets or numbers, spacing, and alignment.

Common Properties:

- **list-style-type**: Defines the type of marker (bullet or number).
  - Values: disc, circle, square, decimal, none.
- **list-style-position**: Controls whether the marker appears inside or outside the list item.
  - Values: inside, outside.
- **list-style-image**: Specifies an image to be used as the list item marker.

Example:

```css
ul {
   list-style-type: square; /* Square bullet points */
   padding-left: 20px;      /* Indentation */
  }

ol {
   list -style-type: decimal; /* Numbered list */
  }

li {
   margin-bottom: 10px; /* spacing between list items */
  }
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Flexbox Example</title>
      <link rel="stylesheet" herf="./style.css">
</head>
<body>

<ul>
      <li>1</li>
      <li>2</li>
      <li>3</li>
</ul>
<ol>
      <li>HTML</li>
      <li>CSS</li>
      <li>JS</li>
</ol>
</body>
</html>
```

- 1
- 2
- 3

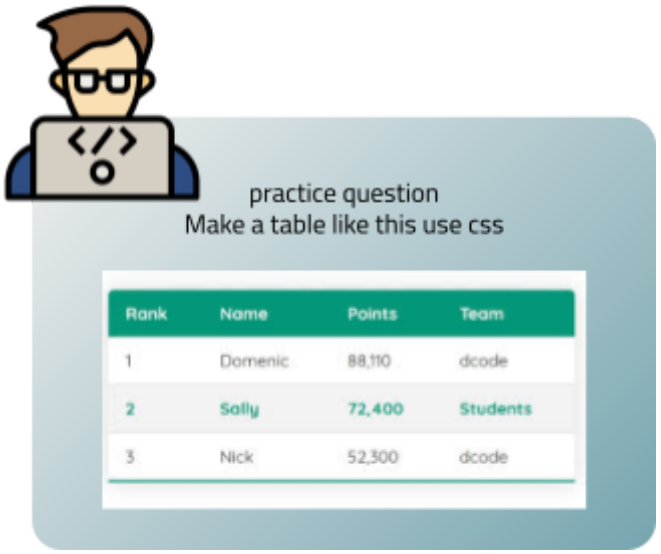
1. HTML
2. CSS
3. JS

9. Table Styling in CSS

Tables are styled using the <table>, <tr>, <th>, and <td> elements. You can control table borders, spacing, alignment, and padding.

Common Properties:

- border: Sets the border around the table, cells, or rows.
- border-collapse: Controls whether table borders are separated or collapsed into a single border.
    - Values: collapse, separate.
- padding: Adds space within cells.
- text-align: Aligns text inside cells (e.g., left, center, right).
- vertical-align: Aligns content vertically in cells (e.g., top, middle, bottom).



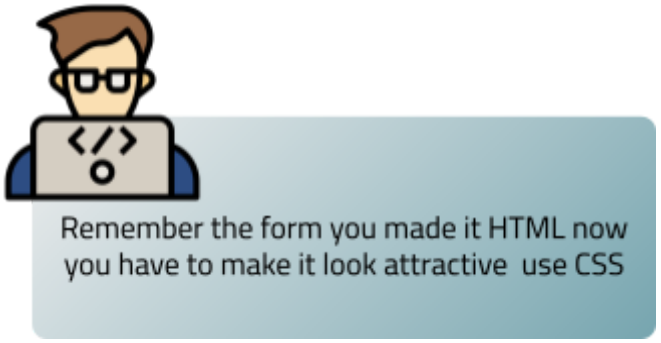
10. Form Styling in CSS

A form in web development refers to a section of a web page designed to collect user input. It typically consists of various input fields, such as text boxes, checkboxes, radio buttons, and dropdown menus, where users can enter data or make selections. This data can then be submitted to a server for processing, such as signing up for an account, submitting a search query, or completing a survey. Forms are essential for interacting with users and capturing information on websites.

Forms can be styled using the <form>, <input>, <textarea>, <select>, <button>, and other form elements. You can control form layout, spacing, and element appearance.

Common Properties:

- **width**: Controls the width of form elements.
- **padding**: Adds space inside input fields or buttons.
- **border**: Defines the border of input fields or buttons.
- **background-color**: Sets the background color of input fields or buttons.
- **text-align**: Aligns text within input fields.



11. Positioning in CSS (Short Explanation)

The **position** property in CSS is used to control how an element is positioned within its parent container or relative to other elements. It defines the positioning context of an element and interacts with other layout properties like top, right, bottom, and left.

Here are the different values for the position property.

1. static (default)

- **Behavior**: The default position for all elements. Elements are positioned according to the normal document flow (top to bottom, left to right).
- **Effect of top, right, bottom, left**: These properties do not affect the element.

2. relative

- **Behavior**: The element is positioned relative to its normal position (where it would be if position were static).
- **Effect of top, right, bottom, left**: These properties shift the element from its original position without affecting the layout of other elements.

3. absolute

- **Behavior**: The element is positioned relative to the nearest **positioned ancestor** (an ancestor element with a position value other than static). If there is no such ancestor, it is positioned relative to the initial containing block (usually the <html> element).
- **Effect of top, right, bottom, left**: These properties move the element relative to its nearest positioned ancestor.

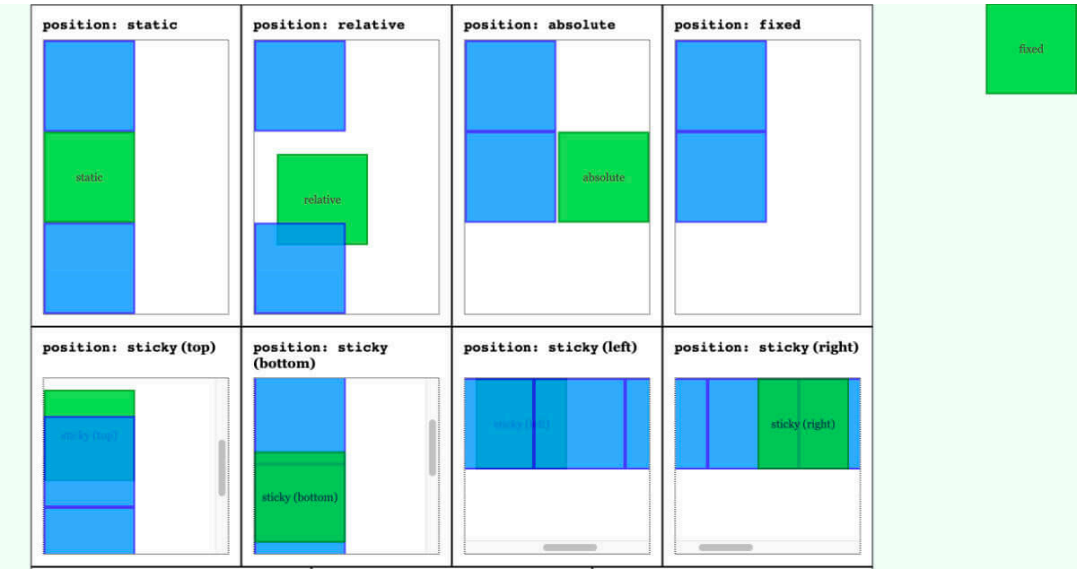
	What is the difference between relative and absolute positioning in CSS? How would you use each?

4. fixed

- **Behavior**: The element is positioned relative to the **viewport** (the browser window), so it stays fixed even when the page is scrolled.
- **Effect of top, right, bottom, left**: These properties position the element relative to the viewport.

5. sticky

- **Behavior**: The element is treated as relative until it reaches a defined scroll position, after which it behaves like fixed. It "sticks" in place when the page is scrolled to a certain point.
- **Effect of top, right, bottom, left**: These properties specify the point at which the element should become sticky.

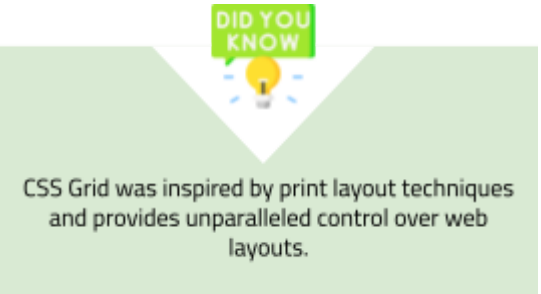


What is z-index?

Explain the difference between z-index and position in CSS. In what scenarios would you need to use z-index?

12. Basic Concepts of CSS Grid

CSS Grid is a layout system that enables the creation of complex, two-dimensional designs using rows and columns. It allows precise control over element placement within a grid container, making it easier to build responsive and structured web layouts



1. **Grid Container**:
    - The element that holds the grid layout is called the **grid container**.
    - You define a grid container using the CSS property display: grid;.
2. **Grid Items**:
    - The direct child elements of the grid container are called **grid items**. They are automatically placed within the grid according to the layout defined.

Key Properties for Grid Layout

1. **display: grid;**

   This property turns an element into a grid container, allowing its children to be grid items.
2. **grid-template-columns**

   Defines the size of columns in the grid. You can specify multiple column sizes, separated by spaces.

13. Basic Concepts of CSS Flex

In CSS, **Flexbox** (short for **Flexible Box Layout**) is a layout model that allows you to design complex layouts with ease, even when the sizes of elements are unknown or dynamic. Flexbox provides an efficient way to distribute space and align items within a container, both horizontally and vertically, without using floats or positioning.

Flexbox

It's like your webpage hired a personal trainer—everything gets aligned, stretched, or shrunk into shape, and no excuses are allowed.

Key Concepts of Flexbox

1. **Flex Container** The parent element that holds the flex items. You need to set the display property to flex or inline-flex on the container to enable Flexbox.
2. **Flex Items** The children of the flex container become flex items. By default, these items are laid out in a row (horizontally), but you can change that using properties like flex-direction.

Main Flexbox Properties

1. flex-direction

Defines the direction of the flex container's children (the flex items). The default value is row, which arranges items horizontally. Other possible values are:

row	Items are placed horizontally, left to right (default).
row-reverse	Items are placed horizontally, right to left.
column	Items are placed vertically, top to bottom.
column-reverse	Items are placed vertically, bottom to top.
border	Sets the style, width, and color of the element's border.

2. justify-content

Aligns the flex items along the main axis (the direction defined by flex-direction).

flex-start	Aligns items to the start of the container.
flex-end	Aligns items to the end of the container.
center	Aligns items in the center of the container.
space-between	Distributes items evenly, with the first item at the start and the last at the end.
space-around	Distributes items evenly with equal space around them.
space-evenly	Distributes items with equal space between them.

3. align-items

Aligns flex items along the cross axis (perpendicular to the main axis).

stretch	Stretches the items to fill the container (default).
flex-start	Aligns items to the start of the cross axis.
flex-end	Aligns items to the end of the cross axis.
center	Aligns items in the center of the cross axis.
baseline	Aligns items along their baseline.

4. align-self

Allows an individual flex item to override the align-items value. This can be useful if you want one item to align differently from the others along the cross axis.

stretch	The element is positioned to fit the container
center	The element is positioned at the center of the container
flex-start	The element is positioned at the beginning of the container
flex-end	The element is positioned at the end of the container

baseline	The element is positioned at the baseline of the container

5. flex-wrap

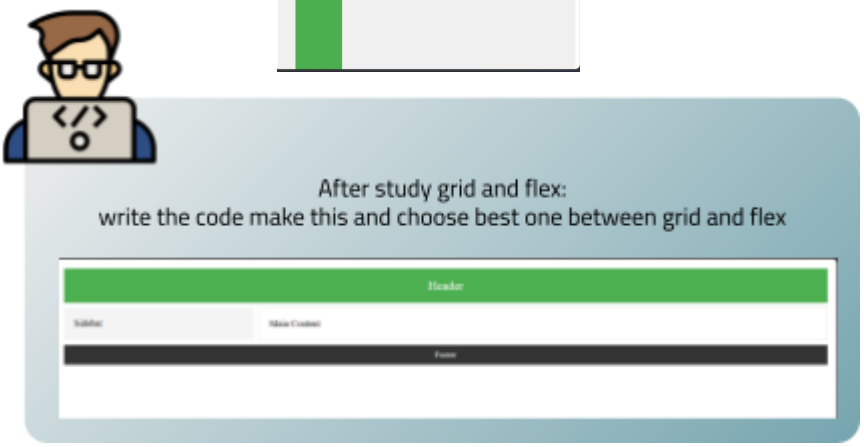
Defines whether the flex items should wrap onto multiple lines if there is insufficient space.

nowrap	Items will not wrap (default).
wrap	Items will wrap to the next line.
wrap-reverse	Items will wrap in reverse order.

6. flex

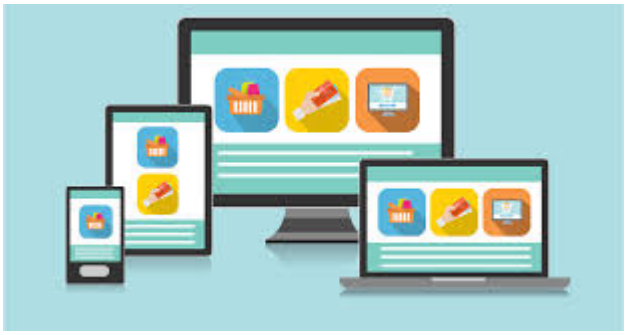
A shorthand for flex-grow, flex-shrink, and flex-basis. It defines how a flex item should grow or shrink to fit the available space.

flex-grow	Defines how much a flex item will grow relative to other items.
flex-shrink	Defines how much a flex item will shrink relative to other items.
flex-basis	Defines the initial size of a flex item before any available space is distributed.



14. Responsive

In CSS, **responsive design** refers to creating web pages that adapt to different screen sizes and devices, ensuring that the user experience is optimal across desktops, tablets, and mobile phones. This is typically achieved through a combination of flexible layouts, media queries, and other CSS techniques.



how to achieve responsive design in CSS:

Media Queries

```css
/* Default styles (mobile-first) */
body {
   font-size: 16px;
  }

/* For larger screens like tablets and desktops */
@media screen and (min-width: 768px) {
      body {
      font-size: 18px;
      }
}
/* For very large screens like tablets and desktops */
@media screen and (min-width: 1200px) {
      body {
      font-size: 20px;
      }
}
```

In the example above:

- **Mobile-first approach**: Styles are written for small screens (phones), and additional styles are added as the screen size increases.
- **min-width**: This media query applies the styles for screens that are at least 768px wide (tablets).

15. Ways to name selector

When naming selectors in CSS, there are several widely used conventions that can help keep your code consistent, readable, and scalable. These naming conventions are particularly useful when you're working on larger projects or teams, as they help to avoid confusion and naming conflicts. Some of the most popular naming conventions are **Camel Case**, **Pascal Case**, and **BEM**. Let's explore each of them.

15.1 Camel Case

Camel case is a naming convention where the first word is written in lowercase, and each The subsequent word begins with an uppercase letter, with no spaces or hyphens between the words.

Example:

- .myButton (first word is lowercase, subsequent words start with uppercase letters)
- .mainContainer
- .activeUser

When to use?

- **JavaScript and CSS-in-JS**: Camel case is commonly used in JavaScript, especially for variables and function names, and it often extends into CSS-in-JS libraries (e.g., styled-components).
- **Single-word class names**: If your class names are short and simple, camel case can be useful for clarity.

Advantages:

- It avoids hyphens or underscores, making it concise.
- Preferred for JavaScript-related code, aligning with common conventions in JS functions and variables.

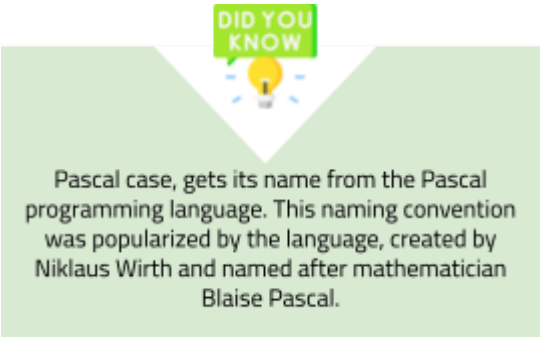
```css
.mybutton{
   background-color: blue;
   color: white;
  }

.mainContainer{
   width: 100%;
   margin: 0 auto;
  }
```

15.2. Pascal Case

Pascal case is similar to camel case, but with **the first letter of each word capitalized**, including the first word. This is often used for naming components, especially in object-oriented programming.

Example:

- .MyButton
- .MainContainer
- .ActiveUser



When to use?

- **Component-based CSS or Frameworks**: Pascal case is often used when naming components in JavaScript or frameworks like React (for React component names). It's less common in plain CSS, but can be used in specific cases like naming larger, modular components.

Advantages:

- Works well when you want to distinguish between regular classes and components.
- Typically used when dealing with libraries or frameworks that treat CSS as components.

```css
.MyButton{
   background-color: red;
   color: white;
  }

.MainContainer{
   width: 100%;
   padding: 20px;
  }
```

15.3 BEM (Block, Element, Modifier)

BEM is a methodology for naming classes that helps to create scalable and maintainable CSS. It stands for **Block**, **Element**, and **Modifier**. It's designed to create modular and reusable components.

BEM Structure:

Block	The main component or wrapper (e.g., .menu, .header).
Element	A part of a block, usually nested or dependent on the block (e.g., .menu__item, .header__title).
Modifier	A state or variation of a block or element (e.g., .menu__item--active, .header__title--large).

Syntax:

- **Block**: .block
- **Element**: .block__element
- **Modifier**: .block__element--modifier

Example:

- .menu: The block represents a menu.
- .menu__item: An item inside the menu (element).
- .menu__item--active: An active menu item (modifier).

When to use?

- **Large-scale projects**: BEM is extremely useful for large projects with multiple developers because it provides a clear and consistent naming convention.
- **Modular and reusable components**: It helps to create small, reusable components that can be easily maintained and extended.

Advantages:

- **Scalable**: BEM is great for maintaining large codebases and helps avoid name conflicts.
- **Readable**: The naming convention makes it clear what each class does and which component it belongs to.

```
/* Block */
.menu{
   display: flex;
  }
/* Element*/
.menu__item{
   padding: 10px;
   color: blue;
  }
/* Modifier*/
.menu__item__active{
   background-color: red;
  }
.menu__item__disabled{
   color: gray;
  }
```

Think like a developer

When working with CSS, it's essential to combine creativity with problem-solving to bring your designs to life. CSS is the tool that transforms plain HTML into visually appealing and interactive web pages. As you style your websites, think about how your choices will impact the user experience.

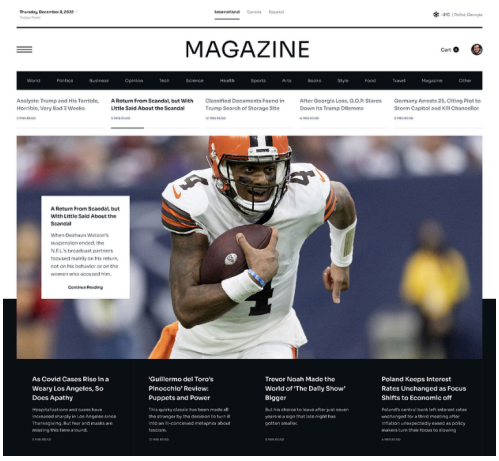
How can you use CSS properties like colors, fonts, and layouts to create a design that aligns with your website's purpose and message?
Consider accessibility—are your font sizes, contrasts, and animations suitable for all users, including those with visual impairments?
As you write CSS, think about how your styles will adapt across various devices and screen sizes. Are you leveraging responsive design techniques like media queries and flexible units to ensure consistency?
How can you structure your CSS code to be reusable and maintainable? Are you using variables or organizing your styles efficiently to make updates easier?
Will your animations and transitions enhance user engagement, or might they be overwhelming or distracting?

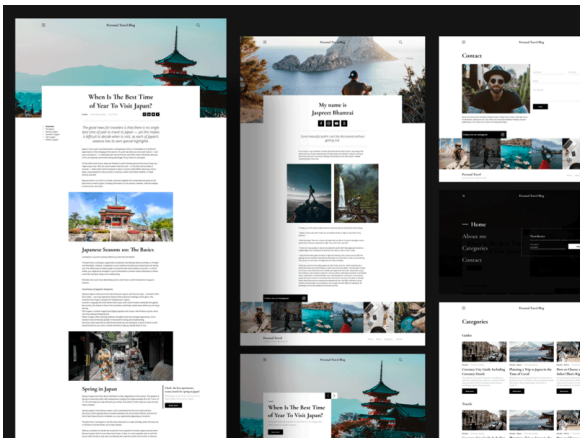
By approaching CSS with a thoughtful and strategic mindset, you can create designs that are not only visually stunning but also functional, accessible, and adaptable to the ever-changing digital landscape.

Capstone Project

Project Overview:

Now you can design your own blog and make it responsive on all devices so that your friends can read it.





Project Requirements

Requirements:

1.use colors , font , formatting .

2. layout use Grid , Flex

3.units px , rem ,%

4.SEO , meta

5.images

6.Responsive

7.BEM

Don't forget to revisit the self-evaluation page at the beginning of the chapter to assess your progress and maximize your learning.

Basic Syntax

```
selector {
  property: value;
}
```

Box Model

```
div {  width: 100px;
       height: 100px;
       padding: 10px;
       border: 5px solid black;
       margin: 20px; }
```

```
Total width = width + padding + border +
margin
```

Text

```
p {  color: #333;
  font-size: 16px;
  font-family: Arial, sans-serif;
  line-height: 1.5;
  text-align: center;}
```

Background

```
div {  background-color: #f4f4f4;
       background-image: url('image.jpg');
       background-size: cover;
       background-repeat: no-repeat;}
```

Selectors

- Universal Selector

```
{color: red;}
```

- Type Selector

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

- Class Selector

```
.example {background-color: yellow;}
```

- ID Selector

```
#header {font-weight: bold;}
```

- Attribute Selector

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

- Descendant Selector (Space)

```
div p {color: green;}
```

- Child Selector (>)

```
div > p {margin-top: 10px;}
```

- Adjacent Sibling Selector (+)

```
h1 + p {font-size: 18px;}
```

- General Sibling Selector (~)

```
h1 ~ p {color: blue;}
```

- Pseudo-classes

```
a:hover {text-decoration: underline;}
```

- Pseudo-elements

```
p::first-letter {font-size: 2em;}
```

Flexbox

```
.container {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: row;
  gap: 10px;
}
```

Grid

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-gap: 20px;
}
.item {
  grid-column: span 2;
}
```

Positioning

- Static (default)

```
div {position: static;}
```

- Relative

```
div {position: relative;
     top: 10px;
     left: 18px;}
```

- Absolute

```
div {position: absolute;
     top: 50px;
     left: 100px;}
```

- Fixed

```
div {position: fixed;
     bottom: 0;
     right: 0;}
```

Responsive Design

```
@media (max-width: 768px) {
  body {
    background-color: lightgray;}
}
```

Borders

```
div {
  border: 2px solid #000;
  border-radius: 10px;
}
```

Transitions

```
button {
  transition: background-color 0.3s ease;
}
button:hover {
  background-color: #555;
}
```

Animations

```
@keyframes slide {
  from {
    transform: translateX(0);
  }
  to {
    transform: translateX(100px);
  }
}
div {
  animation: slide 2s infinite;
}
```

CSS Variables

```
:root {
  --main-color: #3498db;
  --padding: 10px;
}
div {
  background-color: var(--main-color);
  padding: var(--padding);
}
```

Clipping and Shapes

```
div {
 clip-path: circle(50% at 50% 50%);
}
```

Useful Units

- **Absolute:** px, cm, mm
- **Relative:** em, rem, %
- **Viewport:** vw, vh

Colors

- Named Colors

red, blue, green

- HEX

#ff0000

- RGB

rgb(255, 0, 0)

- HSL

hsl(0, 100%, 50%)

- Transparency

rgba(255, 0, 0, 0.5)

Check points

Check Point	Your Answer
What does a CSS rule consist of?	
What are the main types of CSS selectors?	
What is the purpose of the box model?	
When should you use Flexbox?	
What is the difference between em and rem?	
How do you make a webpage responsive?	
What property positions an element in a fixed spot on the screen?	
How do you add a transition effect to an element?	
What is the purpose of CSS variables?	
What is the difference between RGB and HSL?	
Are you writing clean and maintainable CSS, using tools like BEM naming conventions?	
Can you create a fully styled blog or project that is responsive, uses Flexbox or Grid, and includes text, images, and navigation?	

Glossary

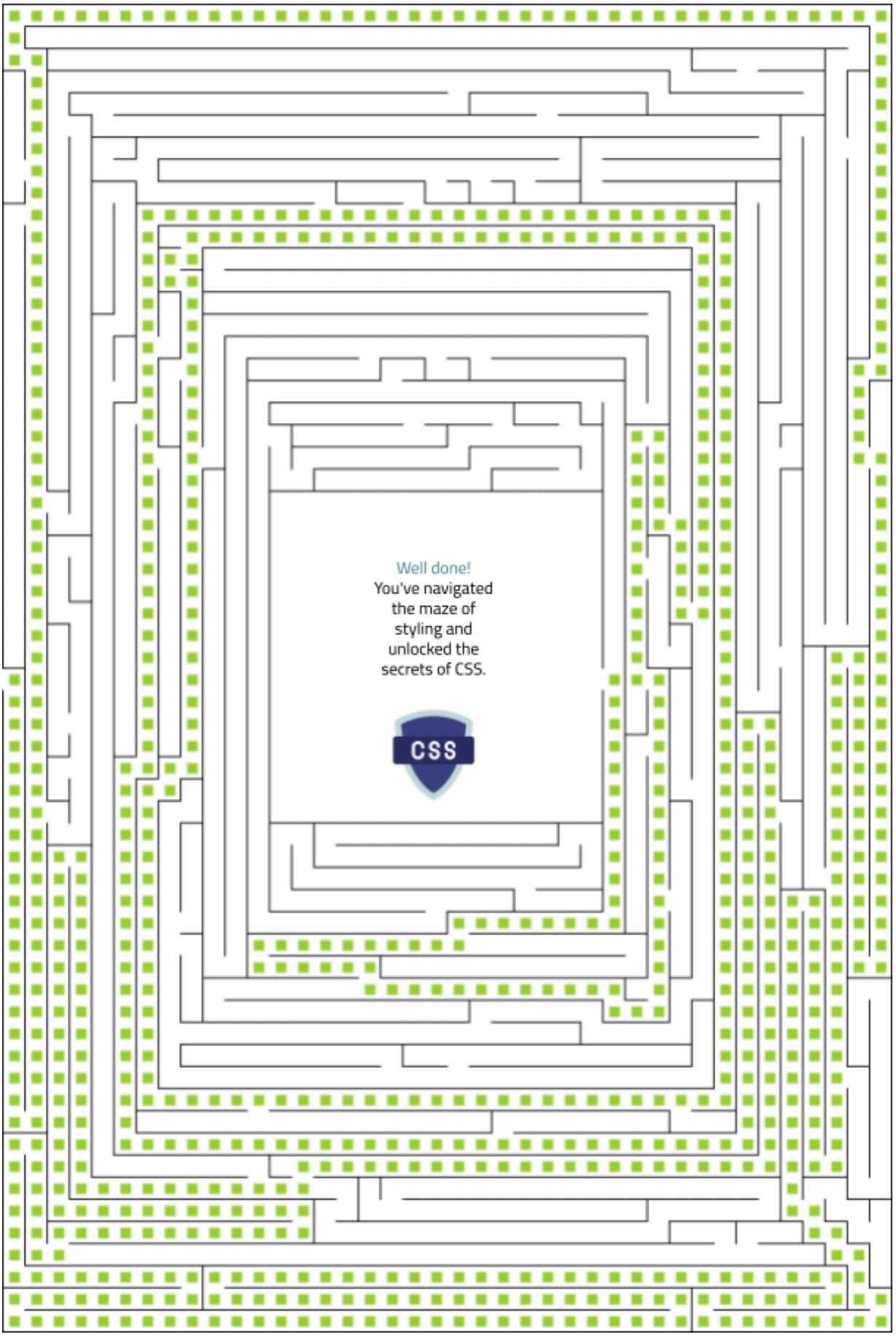
1. **CSS (Cascading Style Sheets):** A stylesheet language used to control the presentation, layout, and styling of HTML elements on a webpage.
2. **Selector:**A part of a CSS rule that identifies which HTML elements the styles will apply to. Examples include type selectors (p), class selectors (.class), and ID selectors (#id).
3. **Property:** The aspect of an element you want to style, such as color, font-size, or margin.
4. **Value:** The setting you assign to a property. For example, red in color: red;.
5. **Declaration:** A combination of a property and a value, defining a specific style. Example: color: blue;.
6. **Rule:** A complete CSS statement consisting of a selector and a set of declarations.
7. **Box Model:** The concept that every HTML element is treated as a rectangular box consisting of the following layers: content, padding, border, and margin.
8. **Pseudo-Class:** A special keyword added to a selector to style elements in a specific state, such as :hover or :nth-child(2).
9. **Pseudo-Element:** A selector that styles specific parts of an element, such as ::before or ::first-letter.
10. **Media Query:** A feature of CSS used to apply styles based on conditions like screen size, resolution, or orientation.
11. **Flexbox:** A CSS layout model designed to provide flexibility in aligning and distributing elements within a container, even when their size is dynamic.
12. **Grid:** A CSS layout system for creating complex, two-dimensional layouts using rows and columns.
13. **Responsive Design:** An approach to web design that ensures a site looks good on all devices by using flexible layouts, media queries, and relative units.
14. **Color Models:** Methods for specifying colors in CSS:
    - **HEX:** A six-character code starting with #, e.g., #ff0000.
    - **RGB:** A method based on red, green, and blue values, e.g., rgb(255, 0, 0).

- **HSL:** A model based on hue, saturation, and lightness, e.g., hsl(0, 100%, 50%).

15. **Opacity:** A property that controls the transparency of an element. Example: opacity: 0.5;.
16. **Transition:** A property that defines the duration of changes between states. Example:
17. **Animation:** A CSS feature used to create moving elements by defining keyframes.
18. **Variables (Custom Properties):** User-defined values that can be reused throughout the stylesheet, declared with --.
19. **Clipping (clip-path):** A property used to define a visible area for an element, hiding everything outside the defined shape.
20. **Z-Index:** A property that specifies the stack order of elements. Higher values appear on top. Example: z-index: 10;.
21. **Positioning:** A method for controlling the placement of elements. Common values are static, relative, absolute, fixed, and sticky.
22. **Units:**
    - **Absolute Units:** Fixed sizes like px (pixels) or cm (centimeters).
    - **Relative Units:** Sizes relative to another value, such as em, rem, or %.
    - **Viewport Units:** Based on the size of the viewport, e.g., vw (viewport width) and vh (viewport height).
23. **Overflow:** Determines how content that overflows an element's box is handled. Common values are visible, hidden, and scroll.
24. **Inline Style:** CSS written directly within an HTML element's style attribute.
25. **External Style Sheet:** CSS stored in a separate file and linked to an HTML document.

CSS MAZE


Well done!
You've navigated the maze of styling and unlocked the secrets of CSS.
CSS


Well done!
You've navigated
the maze of
styling and
unlocked the
secrets of CSS.
CSS

Contact Us

00962788482211

shaiforai

SHAI 2025