Web Technology
CSS(Cascading Style Sheets)

- CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation of a document written in HTML or XML. It controls the layout, colors, fonts, and overall appearance of web pages.
- Separates content (HTML) from presentation (CSS) for better design flexibility and easier maintenance.
- Style multiple pages with a single stylesheet.
- Provides precise control over layout and design.
- Supports responsive design for different devices.
- Bootstrap is CSS framework.

CSS Syntax and Structure

The syntax of CSS consists of selectors, properties, and values.

General Syntax:	**Example:**

selector {
property: value;
}
	p {
	color: blue; /* Sets text color to blue */
	font-size: 16px; /* Sets font size to 16 pixels */
	}

Components:

1. **Selector:** Identifies the HTML element(s) to style.
2. **Property:** Defines the aspect of the element to style (e.g., color, font-size).
3. **Value:** Specifies the style to apply (e.g., red, 16px).

Types of CSS

CSS can be applied in three ways: Inline, Internal, and External.

Type	Description	Example	Advantages	Disadvantages
Inline CSS	Styles applied directly to an HTML element using the style attribute.	Text	Quick for small changes. No need for external files.	Difficult to maintain Cannot reuse styles.
Internal CSS	Styles defined within a <style> tag inside the <head> section of an HTML document.	<style>p { color: green; }</style>	Useful for single-page styling. Keeps styles separate from content.	Not reusable. Increases page size.
External CSS	Styles written in a separate .css file and linked to the HTML document using a <link> tag.	<link rel="stylesheet" href="styles.css">	Reusable across multiple pages. Easier to maintain and update.	Requires an additional file. May delay loading.

CSS Advantages and Applications

Advantages:

1. **Separation of Content and Design:** Keeps HTML clean and focused on structure.
2. **Reusability:** External stylesheets can be reused across multiple web pages.
3. **Consistency:** Ensures uniform design across a website.
4. **Device Responsiveness:** Supports media queries to create responsive designs.
5. **Improved Performance:** External stylesheets reduce redundancy and enhance page loading speed.
6. **Customizability:** Allows granular control over the design of each element.

Applications:

1. **Web Design:** Create visually appealing websites with consistent layouts.
2. **Responsive Design:** Optimize websites for mobile, tablet, and desktop views.
3. **Animations and Transitions:** Add interactivity and smooth effects.
4. **Custom Themes:** Develop unique themes for web applications and content management systems.
5. **Accessibility:** Enhance readability and usability for all users, including those with disabilities.

CSS selectors

A CSS selector is a pattern used to select and target HTML elements for styling. It defines which elements in the HTML document the CSS rules will app.

Selector	Description	Example	Explanation
Universal Selector (*)	Selects all elements on the page.	* { margin: 0; padding: 0; }	Applies a reset by removing default margin and padding from all elements.
Type Selector	Selects all elements of a specific type (e.g., <p>, <h1>).	p { color: blue; }	Styles all <p> elements with blue text.
Class Selector (.classname)	Selects all elements with a specific class.	.highlight { background-color: yellow; }	Applies a yellow background to all elements with the class highlight.
ID Selector (#id)	Selects a single element with a specific ID.	#header { font-size: 24px; }	Sets the font size of the element with the ID header to 24px.
Group Selector (,)	Applies the same styles to multiple selectors.	h1, h2, p { color: green; }	Sets the text color of all <h1>, <h2>, and <p> elements to green.
Descendant Selector (A B)	Selects elements that are descendants of a specified element.	div p { color: red; }	Styles <p> elements inside elements with red text.
Child Selector (A > B)	Selects direct child elements of a specified element.	ul > li { list-style: none; }	Removes bullet points from elements that are direct children of .
Adjacent Sibling Selector (A + B)	Selects the first sibling element immediately following another element.	h1 + p { font-size: 14px; }	Styles the first <p> element immediately following an <h1> element.
General Sibling Selector (A ~ B)	Selects all sibling elements after a specified element.	h1 ~ p { color: gray; }	Styles all <p> elements that are siblings of an <h1> element.
Attribute Selector ([attr])	Selects elements with a specific attribute.	input[type="text"] { border: 1px solid black; }	Styles <input> elements with the type="text" attribute.
Attribute Selector ([attr=value])	Selects elements with a specific attribute and value.	a[target="_blank"] { color: blue; }	Styles <a> elements that open links in a new tab (target="_blank") with blue text.
Attribute Selector ([attr^=value])	Selects elements whose attribute value starts with a specific string.	input[name^="user"] { background-color: lightgray; }	Styles <input> elements whose name attribute starts with "user".
Attribute Selector ([attr$=value])	Selects elements whose attribute value ends with a specific string.	img[src$=".jpg"] { border: 2px solid black; }	Adds a border to <img> elements whose src attribute ends with ".jpg".
Attribute Selector ([attr*=value])	Selects elements whose attribute value contains a specific substring.	a[href*="example"] { color: orange; }	Styles <a> elements with an href containing "example".
Pseudo-class (:hover)	Styles an element when hovered over by the mouse.	a:hover { color: red; }	Changes the color of links to red when hovered over.
Pseudo-class (:nth-child())	Selects elements based on their position among siblings.	li:nth-child(odd) { background-color: lightblue; }	Styles odd-numbered elements with a light blue background.
Pseudo-class (:first-child)	Selects the first child of a parent element.	p:first-child { font-weight: bold; }	Makes the first <p> child of any parent bold.
Pseudo-class (:last-child)	Selects the last child of a parent element.	p:last-child { color: green; }	Changes the text color of the last <p> child of any parent to green.
Pseudo-element (::before)	Inserts content before the content of an element.	h1::before { content: "★ "; color: gold; }	Adds a gold star before the text of all <h1> elements.

Pseudo-element (::after)	Inserts content after the content of an element.	h1::after { content: " ★"; color: gold; }	Adds a gold star after the text of all <h1> elements.
Pseudo-element (::first-letter)	Styles the first letter of an element.	p::first-letter { font-size: 24px; color: red; }	Makes the first letter of all <p> elements larger and red.
Pseudo-element (::first-line)	Styles the first line of an element.	p::first-line { font-weight: bold; }	Makes the first line of all <p> elements bold.
Not Selector (:not())	Excludes elements that match a specific selector.	p:not(.highlight) { color: gray; }	Styles all <p> elements that do not have the class highlight with gray text.

Difference Between Pseudo-classes and Pseudo-elements

Aspect	Pseudo-classes	Pseudo-elements
Purpose	A **pseudo-class** in CSS defines a special state of an element. It allows you to apply styles to elements based on their state, position in the DOM, or user interactions without requiring additional classes or IDs in the HTML.	A **pseudo-element** in CSS is used to style specific parts of an element or add content dynamically. It allows targeting portions of an element, such as the first letter, first line, or inserting content before or after the element.
Syntax	Single colon (:) is used before the pseudo-class name.	Double colon (::) is used before the pseudo-element name (single colon is also supported for older CSS).
Behavior	Applies styles dynamically based on user interaction or state.	Applies styles to a specific part of the element or adds decorative content.
Dynamic Content	Cannot add new content; only styles existing elements.	Can add content dynamically using the content property (e.g., ::before, ::after).
Examples	:hover, :focus, :nth-child, :visited.	::before, ::after, ::first-line, ::selection.

CSS Pseudo-classes

Pseudo-class	Description	Example	Explanation
:hover	Styles an element when the mouse hovers over it.	a:hover { color: red; }	Changes the color of links to red when hovered over by the mouse.
:focus	Styles an element when it gains focus (e.g., input fields).	input:focus { border: 2px solid blue; }	Adds a blue border to input fields when they are focused.
:active	Styles an element while it is being activated (e.g., clicked).	button:active { background-color: gray; }	Changes the background color of buttons to gray while they are clicked.
:visited	Styles links that have already been visited.	a:visited { color: purple; }	Changes the color of visited links to purple.
:nth-child(n)	Selects elements based on their position among siblings (1-based index).	li:nth-child(odd) { background-color: lightblue; }	Styles odd-numbered elements with a light blue background.
:nth-of-type(n)	Selects elements of a specific type based on their position among siblings.	p:nth-of-type(2) { font-size: 20px; }	Styles the second <p> element of each parent with a larger font size.
:first-child	Selects the first child of a parent element.	p:first-child { font-weight: bold; }	Makes the first <p> child of any parent bold.

:last-child	Selects the last child of a parent element.	p:last-child { color: green; }	Changes the text color of the last <p> child of any parent to green.
:only-child	Selects an element if it is the only child of its parent.	div:only-child { border: 1px solid black; }	Adds a border to a element if it is the only child of its parent.
:not(selector)	Excludes elements that match the given selector.	p:not(.highlight) { color: gray; }	Styles all <p> elements except those with the class highlight in gray.
:empty	Selects elements that have no children (including text nodes).	div:empty { display: none; }	Hides empty elements.
:checked	Selects checkboxes or radio buttons that are checked.	input:checked { background-color: yellow; }	Highlights checked input elements with a yellow background.
:disabled	Selects elements that are disabled.	button:disabled { opacity: 0.5; }	Makes disabled buttons semi-transparent.
:enabled	Selects elements that are enabled.	input:enabled { border: 1px solid green; }	Styles enabled input elements with a green border.
:first-of-type	Selects the first element of a specific type within a parent.	p:first-of-type { font-style: italic; }	Makes the first <p> element of each parent italicized.
:last-of-type	Selects the last element of a specific type within a parent.	p:last-of-type { color: red; }	Changes the text color of the last <p> element of each parent to red.
:nth-last-child(n)	Selects elements based on their position from the end among siblings.	li:nth-last-child(2) { font-weight: bold; }	Makes the second-to-last element bold.
:nth-last-of-type(n)	Selects elements of a specific type based on their position from the end among siblings.	p:nth-last-of-type(1) { color: blue; }	Styles the last <p> element of each parent with blue text.
:root	Selects the root element of the document (usually <html>).	:root { --main-color: #333; }	Defines a CSS variable --main-color at the root level.
:lang(language)	Selects elements with a specific language attribute.	p:lang(en) { font-style: italic; }	Styles <p> elements with a lang="en" attribute in italic.

CSS Pseudo-elements

Pseudo-element	Description	Example	Explanation
::before	Inserts content before the content of an element.	h1::before { content: "★ "; color: gold; }	Adds a gold star before the text of all <h1> elements.
::after	Inserts content after the content of an element.	h1::after { content: " ★"; color: gold; }	Adds a gold star after the text of all <h1> elements.
::first-letter	Styles the first letter of an element.	p::first-letter { font-size: 24px; color: red; }	Makes the first letter of all <p> elements larger and red.
::first-line	Styles the first line of an element.	p::first-line { font-weight: bold; }	Makes the first line of all <p> elements bold.
::selection	Styles the portion of an element that is selected by the user.	::selection { background-color: yellow; color: black; }	Changes the background and text color of selected text to yellow and black, respectively.
::placeholder	Styles the placeholder text of an input element.	input::placeholder { color: gray; font-style: italic; }	Makes placeholder text gray and italicized.

::marker	Styles the marker (bullet or number) of list items.	li::marker { color: blue; font-size: 18px; }	Changes the color and size of list markers to blue and 18px.
::backdrop	Styles the backdrop of elements in full-screen mode.	::backdrop { background-color: rgba(0, 0, 0, 0.5); }	Adds a semi-transparent black background to the backdrop of full-screen elements.

CSS Box Model

- The CSS Box Model is a fundamental concept in web design and layout.
- It describes how every HTML element is represented as a rectangular box consisting of four areas: content, padding, border, and margin.
- Understanding the box model is essential for controlling the spacing and layout of elements on a webpage.

Components of the Box Model

Component	Description	Purpose	
Content	The innermost area where the actual content (text, images, etc.) is displayed. The size of the content area can be controlled using width and height properties.	Defines the size of the element's content.	div { width: 200px; height: 100px; }
Padding	The space between the content and the border. Itcan be set using the padding property. It accepts individual values for top, right, bottom, and left, or shorthand.	Provides inner spacing around the content, ensuring it doesn't touch the border.	div { padding: 20px; /* Uniform padding on all sides */ padding: 10px 15px; /* Top/Bottom: 10px, Left/Right: 15px */ }
Border	The boundary that wraps around the padding and content. It can be styled using the border property, specifying width, style, and color.	Defines the thickness, style, and color of the box's edge.	div { border: 2px solid black; /* 2px thick solid black border */ border-radius: 10px; /* Rounded corners */ }
Margin	The outermost area that creates space between the element and neighboring elements. It can collapse with adjacent element margins (margin collapsing).	Controls the spacing between the element and other elements on the page.	div { margin: 20px; /* Uniform margin on all sides */ margin: 10px 15px; /* Top/Bottom: 10px, Left/Right: 15px */ }

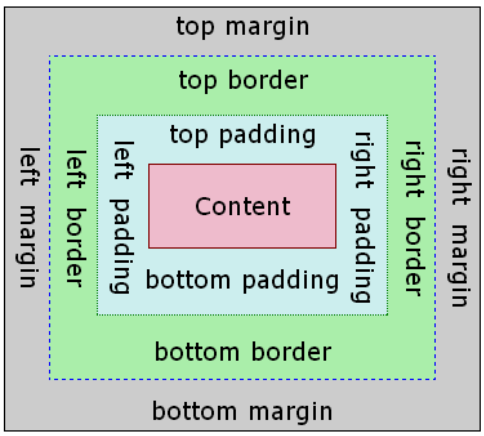
Box Sizing

The box-sizing property determines how the total width and height of an element are calculated:

content-box (default): Width and height include only the content. Padding and border are added outside.

border-box: Width and height include content, padding, and border.

```
div
 { width: 200px;
 height: 100px;
box-sizing: border-box; /* Ensures total size includes padding and border */
}
```



CSS Display Properties

It define how an element is displayed in the document layout (e.g., block, inline, flex, grid).

Property	Description	Example
block	Makes the element a block-level element, occupying the full width of its parent.	Block Element
inline	Makes the element an inline element, occupying only as much width as necessary.	Inline Element
inline-block	Combines block and inline behavior: width/height can be set, but elements remain inline.	Inline Block
none	Hides the element from the layout (not rendered in the document flow).	Hidden
flex	Enables a flex container, allowing children to align and distribute space dynamically.	Flex Container
grid	Enables a grid container, allowing children to be placed in rows and columns.	Grid Container
inline-flex	Similar to flex but behaves like an inline element.	Inline Flex
table	Makes the element behave like a table.	Table Element
table-row	Makes the element behave like a table row.	Table Row
table-cell	Makes the element behave like a table cell.	Table Cell

CSS Positioning

Specifies how an element is positioned in relation to its normal position or its parent (e.g., static, relative, absolute, fixed, sticky).

Property	Description	Example
static	Default positioning; elements are placed in the normal document flow.	Static Position
relative	Positioned relative to its normal position.	Relative Position
absolute	Positioned relative to the nearest positioned ancestor (non-static).	Absolute Position
fixed	Positioned relative to the viewport; stays in place during scrolling.	Fixed Position
sticky	Toggles between relative and fixed based on scroll position.	Sticky Position
z-index	Specifies the stack order of elements. Higher values are in front of lower values.	Stack Order

Float and Clear Properties

It controls the positioning of elements, allowing them to float left or right and clearing space around floated elements. Float aligns elements to the left or right, allowing content to flow around. Clear stops elements from being placed next to floated elements.

Property	Description	Example
float: left	Floats the element to the left, allowing inline content to wrap around it.	
float: right	Floats the element to the right, allowing inline content to wrap around it.	
clear: left	Prevents elements from being positioned next to floated elements on the left.	Clear Left

clear: right	Prevents elements from being positioned next to floated elements on the right.	Clear Right
clear: both	Prevents elements from being positioned next to floated elements on both sides.	Clear Both
clear: none	Default value; allows elements to be positioned next to floated elements.	No Clear

CSS Color Models

Different ways to represent colors in CSS

Color Model	Description	Example
Named Colors	A set of predefined color names (e.g., "red", "blue", "green").	color: red;
HEX	A hexadecimal representation of color, using 6 digits (e.g., #RRGGBB). Each pair represents Red, Green, and Blue values.	color: #ff5733;
RGB	Defines colors using the Red, Green, and Blue components in a range from 0 to 255.	color: rgb(255, 87, 51);
RGBA	Similar to RGB but includes an Alpha value for opacity (0 to 1, where 1 is fully opaque).	color: rgba(255, 87, 51, 0.5);
HSL	Defines color using Hue (0 to 360), Saturation (0% to 100%), and Lightness (0% to 100%).	color: hsl(9, 100%, 60%);
HSLA	Similar to HSL but includes an Alpha value for opacity.	color: hsla(9, 100%, 60%, 0.5);

CSS Background Properties

CSS properties that define the background of an element, including background color, image, position, size, repeat behavior, and attachment.

Property	Description	Example
background-color	Sets the background color of an element.	background-color: #ff5733;
background-image	Sets an image as the background of an element.	background-image: url('image.jpg');
background-position	Defines the position of the background image (e.g., top, bottom, left, right, center, or specific values).	background-position: center;
background-size	Specifies the size of the background image (e.g., cover, contain, or specific dimensions).	background-size: cover;
background-repeat	Defines whether the background image should repeat (e.g., repeat, no-repeat, repeat-x, repeat-y).	background-repeat: no-repeat;
background-attachment	Specifies whether the background image should scroll with the page or be fixed.	background-attachment: fixed;

CSS Flexbox

- Flexbox (Flexible Box Layout) is a CSS layout model that allows you to design complex layouts with ease.
- It provides an efficient way to align and distribute space among items in a container, even when their sizes are unknown or dynamic.
- Flexbox helps in creating responsive designs without using floats or positioning.
- **Flex Container**: The parent element that holds the flex items.
- **Flex Items**: The child elements inside the flex container that are laid out according to the flexbox model.

Properties of Parent (Flex Container)

Property		Description	Example
display: flex		Defines the container as a flex container, enabling flexbox layout.	div { display: flex; }
justify-content	**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. **space-between:** Distributes items evenly with the first item at the start and the last item at the end. **space-around:** Distributes items evenly with equal space around each item.	Aligns the flex items along the main axis (horizontal by default). Options: flex-start, flex-end, center, space-between, space-around.	div { justify-content: center; }
align-items	**flex-start:** Aligns items to the top of the container. **flex-end:** Aligns items to the bottom of the container. **center:** Aligns items in the center vertically. **baseline:** Aligns items based on their baseline. **stretch:** Stretches the items to fill the container (default behavior).	Aligns the flex items along the cross axis (vertical by default). Options: flex-start, flex-end, center, baseline, stretch.	div { align-items: center; }
flex-wrap	**nowrap:** All items are placed in a single line. **wrap:** Items will wrap onto the next line if needed. **wrap-reverse:** Items will wrap onto the next line in reverse order.	Specifies whether the flex items should wrap onto multiple lines if necessary. Options: nowrap, wrap, wrap-reverse.	div { flex-wrap: wrap; }

Properties of Children (Flex Items)

Property	Description	Example
flex-grow	Defines the ability of a flex item to grow relative to the other items. Default is 0 (no growth).	div { flex-grow: 1; }
flex-shrink	Defines the ability of a flex item to shrink relative to the other items. Default is 1 (can shrink).	div { flex-shrink: 1; }
flex-basis	Specifies the initial size of the flex item before any growing or shrinking. Default is auto.	div { flex-basis: 100px; }
align-self	Allows individual flex items to override the align-items property and align themselves differently. Options: auto, flex-start, flex-end, center, baseline, stretch.	div { align-self: flex-end; }

CSS Grid

CSS Grid Layout is a powerful two-dimensional layout system for the web.
It allows you to create complex grid-based designs with ease, controlling both rows and columns.
Unlike Flexbox, which works in a single direction (either row or column), CSS Grid allows you to control both dimensions at once, making it ideal for creating intricate web layouts.
Grid Container: The parent element that holds the grid items.
Grid Items: The child elements inside the grid container that are placed into the grid structure.

Properties of Grid Container

Property	Description	Example
display: grid	Defines the container as a grid container, enabling the grid layout system.	div { display: grid; }
grid-template-rows	Specifies the number and size of rows in the grid. You can define fixed, flexible, or auto-sized rows.	div { grid-template-rows: 100px 200px auto; }
grid-template-columns	Specifies the number and size of columns in the grid. Similar to grid-template-rows.	div { grid-template-columns: 1fr 2fr 1fr; }
gap	Defines the space between grid rows and columns. This property is shorthand for row-gap and column-gap.	div { gap: 10px; }

Properties of Grid Items

Property	Description	Example
grid-row	Defines the start and end position of a grid item along the row axis (vertical).	div { grid-row: 1 / 3; }
grid-column	Defines the start and end position of a grid item along the column axis (horizontal).	div { grid-column: 2 / 4; }

CSS Transitions

CSS Transitions allow you to change property values smoothly (over a given duration) when an element's state changes, like when hovering over an element.

Syntax of CSS Transitions:

transition: property duration timing-function delay;

- **property**: The CSS property to which the transition effect is applied (e.g., color, background-color, width).
- **duration**: How long the transition takes (e.g., 1s, 500ms).
- **timing-function**: Specifies the speed curve of the transition (e.g., ease, linear, ease-in-out).
- **delay**: The delay before the transition starts (optional).

CSS Transformations

CSS Transformations allow you to change the position, size, and shape of an element. You can apply transformations like translate, rotate, scale, and skew to elements.

Types of Transformations:

Transformation	Description	Example
translate	Moves an element from its current position (x, y).	transform: translate(50px, 100px);
rotate	Rotates an element around a point (usually the center).	transform: rotate(45deg);
scale	Resizes an element.	transform: scale(1.5);
skew	Distorts an element by skewing it along the x or y axis.	transform: skew(20deg, 10deg);

CSS Animations

CSS Animations allow you to create more complex animations using @keyframes and animation properties. These are more powerful than transitions because they can run continuously or repeat, and you can define multiple steps in an animation.

Keyframe Syntax:

```
@keyframes animation-name {
  from {
    /* Initial state */
  }     to {
    /* Final state */   } }
```

- **animation-name**: The name of the animation (e.g., slide, fade).
- **animation-duration**: How long the animation lasts (e.g., 2s, 500ms).
- **animation-timing-function**: Specifies the speed curve of the animation (e.g., ease, linear).
- **animation-iteration-count**: Defines how many times the animation should run (e.g., infinite for infinite loop).
- **animation-delay**: Specifies a delay before the animation starts.

Media Queries

Media Queries allow you to apply CSS styles conditionally, based on the characteristics of the device, such as screen width, height, or resolution. They help in creating responsive designs that adjust to different screen sizes.

Syntax of Media Queries:

```
@media (condition) {
  /* CSS rules */
}
```

- **Condition**: The condition or criteria to apply the styles, such as min-width, max-width, orientation, etc.

Responsive Units

Responsive units are used in CSS to create designs that scale smoothly with different screen sizes. These units help make elements flexible and adaptable.

Unit	Description	Example	Use Case
%	Represents a percentage relative to the parent element's size.	width: 50%;	Good for fluid layouts. The element will take up 50% of its parent.
em	Relative to the font-size of the parent element.	font-size: 2em;	Scales with the parent element's font size.
rem	Relative to the root element's font size (usually html).	font-size: 2rem;	Useful for consistent scaling across the entire page.
vh	Relative to 1% of the viewport height.	height: 50vh;	Useful for setting the height relative to the viewport.
vw	Relative to 1% of the viewport width.	width: 50vw;	Useful for setting the width relative to the viewport.