Fonts -

CSS fonts are essential for controlling the appearance and readability of text on web pages. You can choose typefaces, sizes, weights, styles, and even load custom fonts using various CSS properties.

Key Font Properties in CSS

- **font-family:** Sets the typeface(s) used. List multiple fonts for fallback (e.g., "Arial, Helvetica, sans-serif").
- **font-size:** Changes the size of the text. Units include px, em, rem, %, etc.
- **font-weight:** Controls text thickness (normal, bold, or numeric values like 100–900).
- **font-style:** Sets style like normal, italic, or oblique.
- **font-variant:** Commonly used for small-caps.
- **line-height:** Sets space between lines of text.
- **letter-spacing:** Adjusts space between letters.
- **text-transform:** Alters casing (e.g., uppercase, lowercase).

Generic Font Families

All font names in CSS belong to one of these five generic font families:

- **Serif:** Traditional, with small “feet” (e.g., Times New Roman).
- **Sans-serif:** No feet, more modern (e.g., Arial, Helvetica).
- **Monospace:** All letters are the same width (e.g., Courier New).
- **Cursive:** Imitates handwriting.
- **Fantasy:** Decorative or playful fonts.

Using Fonts in CSS

Basic Syntax

```
/* Set preferred font and fallback options */

body {

  font-family: "Roboto", "Arial", sans-serif;

  font-size: 18px;

  font-weight: 400;

  font-style: normal;

}
```

- If "Roboto" is missing, the browser tries "Arial", then defaults to any sans-serif font.

Font Shorthand Property

You can combine many font settings in one line:

```
p {
  font: italic small-caps bold 24px/1.5 "Georgia", serif;
}
```

This sets italic, small-caps, bold, 24px size, 1.5 line height, prefers Georgia, and falls back to serif.

Custom & Web Fonts

- **Google Fonts:** Add a <link> in <head> or use @import, then use its name in font-family.
- **@font-face:** Load your own font files.

```
@font-face {
  font-family: 'MyFont';
  src: url('MyFont.woff2') format('woff2');
}
body { font-family: 'MyFont', sans-serif; }
```

- **Best Practice:** Always declare fallback fonts.

Example: Styling Headers

```
h1 {
  font-family: 'Poppins', Arial, sans-serif;
  font-size: 2.5rem;
  font-weight: 700;
  font-style: italic;
}
```

Tips for Using Fonts

- Limit to 2–3 font families per site for consistency and performance.
- Always include generic fallback families.
- Use readable font sizes and contrast for accessibility.

- Optimize custom fonts to load quickly.

By mastering CSS font properties, you can greatly improve the aesthetics and readability of your website text.

Custom Fonts -

To use **custom fonts** in CSS, you go beyond the standard web-safe fonts and load your own font files (like .woff2, .woff, .ttf, etc.) or use online font services (such as Google Fonts). This lets you use unique typography for your website, matching brand or design needs.

How to Use Custom Fonts in CSS

1. The @font-face Rule

The @font-face at-rule allows you to define custom fonts by specifying a name (for CSS use) and a file URL (or multiple URLs for different font formats).

Example:

```
@font-face {
  font-family: 'MyCustomFont';
  src: url('fonts/MyCustomFont.woff2') format('woff2'),
       url('fonts/MyCustomFont.woff') format('woff');
  font-weight: normal;
  font-style: normal;
}
body {
  font-family: 'MyCustomFont', Arial, sans-serif;
}
```

- font-family: The name you'll use in your CSS.
- src: The path(s) to your font file(s) and their format(s).
- You can add multiple @font-face blocks for different font weights, styles, or unicode ranges.

Tip: Always supply fallback fonts in font-family.

2. Using Google Fonts (or Similar Font Services)

With Google Fonts:

1. Go to fonts.google.com, select a font and styles.
2. Copy the <link> provided by Google into your HTML's <head>, for example:
3. <link href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;700&display=swap" rel="stylesheet">
4. Reference the font in your CSS:
5. body {
6. font-family: 'Open Sans', Arial, sans-serif;
7. }

Advantages: Fast loading, optimized for the web, lots of styles.

3. Best Practices and Notes

- Prefer modern formats: **woff2** is most efficient and widely supported (then woff).
- Host your fonts locally for full control, or use CDN services for ease and speed.
- Always include **fallback** font families in case your custom font fails to load.
- Respect font licensing; only use fonts you're permitted to.

Common Example: Self-Hosted Custom Font

```
@font-face {
  font-family: 'CoolFont';
  src: url('/fonts/CoolFont.woff2') format('woff2');
  font-weight: 400;
  font-style: normal;
}
h1 {
  font-family: 'CoolFont', Georgia, serif;
  font-weight: 400;
}
```

Summary:
Custom fonts make your site stand out. Use @font-face to load fonts from files, or bring in cloud-hosted fonts like Google Fonts by adding a link and referencing in CSS. Always provide fallback fonts and check licensing for any font you use.

Common Font Format -

The most common font file formats used for web and digital projects are:

- **TTF (TrueType Font):**
  One of the oldest and most widely used font formats, originally developed by Apple. TTF fonts offer broad compatibility with both macOS and Windows and are installable on most systems. However, their file size is larger compared to web-optimized formats, and they are not as efficient for web use.
- **OTF (OpenType Font):**
  Built on the TTF format, OTF adds advanced features like ligatures, alternate characters, and greater Unicode support. Suitable for both print and digital use, OTF fonts are popular among designers looking for advanced typographic options.
- **WOFF / WOFF2 (Web Open Font Format):**
  Specifically designed for the web, WOFF and its newer version WOFF2 provide strong compression for smaller file sizes and faster loading. These are the standard formats recommended for modern browsers due to their efficient performance and cross-browser compatibility. WOFF2 offers even greater compression than WOFF.
- **EOT (Embedded OpenType):**
  Developed by Microsoft for older versions of Internet Explorer (IE8 and below). It's used purely for legacy browser support and is largely obsolete today.
- **SVG Fonts:**
  Vector-based font format, mainly for scalable graphics/icons within SVG images. Not commonly used for standard text due to limited browser support and technical drawbacks.

Summary of usage:

- For **websites**: Use **WOFF2** first, with **WOFF** as a fallback for older browsers.
- For **general digital or print work**: Use **OTF** or **TTF**.
- Use **EOT** only if legacy IE8 support is strictly required.
- Use **SVG fonts** for scalable graphics/icons rather than main website text.

This combination ensures broad compatibility, fast loading, and quality typography across the web and other uses.

Text Properties -

CSS provides a wide range of **text properties** to control the appearance and layout of textual content on your website. Here are the most commonly used CSS text properties, along with their purposes:

Common CSS Text Properties

Property	Description	Example Value
color	Sets the color of the text	#333, red, rgb(0,0,0)
text-align	Aligns text within its block container	left, right, center
text-align-last	Sets alignment of the last line in a block	justify, center
text-decoration	Adds decorations like underlines, lines, overlines	underline, line-through
text-decoration-color	Sets the color for text decorations	blue, #f00
text-decoration-line	Specifies type of text decoration	underline, overline
text-decoration-style	Sets the style of the decoration line	dashed, wavy, solid
text-indent	Indents the first line of text	30px, 2em
text-justify	Controls justification method	inter-word, inter-character
text-overflow	How overflowed hidden text is signaled	ellipsis, clip
text-transform	Controls capitalization (case) of the text	uppercase, lowercase
text-shadow	Adds shadow to text	2px 2px 5px gray
letter-spacing	Controls space between characters	0.1em, 2px
line-height	Sets space between lines of text	1.5, 24px

word-spacing	Controls space between words	5px, 1em
direction	Sets text direction	ltr, rtl
white-space	Controls how white space is handled	normal, nowrap, pre
unicode-bidi	Manages the level of embedding for bidi text	normal, embed, bidi-override

Example Usage

```
p {
  color: #333;
  text-align: justify;
  text-indent: 2em;
  letter-spacing: 0.05em;
  line-height: 1.8;
  text-decoration: underline wavy red;
  text-transform: capitalize;
  text-shadow: 1px 1px 4px #aaa;
}
```

These properties allow you to:

- Change the color, size, and spacing of letters and words
- Align text horizontally and vertically
- Decorate text with lines, shadows, and other visual effects
- Transform or capitalize text
- Handle overflow and whitespace for responsive and readable content

You can mix and match these properties to achieve your desired text presentation and enhance your site's readability and style.

Box Model -

The CSS Box Model is a fundamental concept in web development that describes how elements on a webpage are structured and spaced. Every HTML element is considered a rectangular box composed of four parts or layers:

1. **Content**: This is the actual content inside the element, such as text, images, or other media. It is the innermost part of the box.
2. **Padding**: This is the transparent space between the content and the border. It adds space inside the element, around the content.
3. **Border**: Surrounding the padding (or content if padding is zero), the border is a visible line that wraps the element. Its thickness and style can be customized.
4. **Margin**: This is the transparent space outside the border, separating the element from other nearby elements.

The box model essentially defines how much space an element occupies. The total width and height of the element are calculated as:

Total width = content width + padding-left + padding-right + border-left + border-right + margin-left + margin-right
Total height = content height + padding-top + padding-bottom + border-top + border-bottom + margin-top + margin-bottom

By default, CSS uses the **content-box** sizing model, where width and height apply only to the content area, and padding and border add extra size outside of it. With the **border-box** model, the width and height include the padding and border, keeping the total size fixed.

The box model allows for precise control over spacing and layout between elements, enabling designers to create visually appealing and well-structured pages.

Here is a simple example CSS demonstrating the box model:

```
div {
  width: 300px;        /* content width */
  padding: 20px;       /* space inside around content */
  border: 5px solid black; /* border thickness and style */
  margin: 15px;        /* space outside border */
  box-sizing: content-box; /* default model */
}
```

This means the displayed width will be wider than 300px due to padding and border added outside content.

Understanding and manipulating the CSS box model is key to effective layout and design in web development.

CSS Gradients -

CSS gradients are a way to create smooth transitions between two or more colors, which can be used as backgrounds or artistic effects in web design. There are several types of CSS gradients, with the most common being linear gradients.

CSS Linear Gradients

- A **linear gradient** transitions colors progressively along a straight line.
- You define it using the linear-gradient() function in CSS.
- The gradient can flow in any direction: top to bottom (default), left to right, diagonally, or at any angle.

Syntax

```
background-image: linear-gradient(direction, color-stop1, color-stop2, ...);
```

- **direction** (optional): Specifies the direction of the gradient. This can be keywords like to right, to bottom left, or an angle like 45deg.
- **color stops**: At least two colors are required. You can specify more colors for multi-color gradients, and optionally set where each color stop begins (percentage or length).

Examples

- Gradient from top to bottom (default direction):

```
background-image: linear-gradient(red, yellow);
```

- Gradient from left to right:

```
background-image: linear-gradient(to right, red, yellow);
```

- Diagonal gradient from top left to bottom right:

```
background-image: linear-gradient(to bottom right, red, yellow);
```

- Gradient with an angle (90 degrees = left to right):

```
background-image: linear-gradient(90deg, red, yellow);
```

- Multi-color gradient:

```
background-image: linear-gradient(to right, red, orange, yellow, green, blue);
```

- Hard color stops for sharp transitions between colors:

```
background-image: linear-gradient(to right, red 15%, yellow 15%);
```

Other Gradient Types (Briefly)

- **Radial Gradients**: Colors spread out from a center point.
- **Conic Gradients**: Colors rotate around a center like a pie chart.

Usage Notes

- Gradients are often applied via the background-image property.
- You can create repeating gradients with functions like repeating-linear-gradient.
- Colors can be specified using color names, HEX, RGB, RGBA, HSL, etc.

Here is a practical example that you could use in your notes:

```
div {
  height: 200px;
  background-image: linear-gradient(to right, red, orange, yellow, green, blue, indigo, violet);
}
```

This creates a rainbow effect horizontally across the div from left (red) to right (violet).

Understanding CSS gradients helps in creating visually appealing and modern web designs with smooth color transitions without needing images.

Repeating Linear Gradient -

A repeating linear gradient in CSS is a way to create a linear gradient pattern that repeats indefinitely across the background of an element. It uses the repeating-linear-gradient() function, which works similarly to linear-gradient() but repeats the sequence of color stops over and over.

Syntax

```
background-image: repeating-linear-gradient(angle | to side-or-corner, color-stop1, color-stop2, ...);
```

- **angle** or **direction** (optional): Specifies the direction of the gradient line (e.g., 45deg, to right).
- **color stops**: Specifies the colors and where they stop/transition along the gradient line. These stops define the length of the repeating segment.

Key Concept

The length of the repeating pattern is determined by the distance between the first and last color stop. This "segment" is then repeated infinitely across the element's background.

Example: Zebra Stripes

```
background-image: repeating-linear-gradient(
  -45deg,       /* angle direction */
  transparent,
  transparent 20px,
  black 20px,
  black 40px
);
```

This creates a diagonal striped pattern with transparent and black stripes, each 20px wide, repeating infinitely.

Example: Horizontal Bars

```
background-image: repeating-linear-gradient(
  to bottom,
  lightblue,
  lightblue 10%,
  pink 10%,
  pink 20%
);
```

This produces horizontal bars repeating vertically with alternating colors taking 10% of the height each.

Explanation

- The color stops define the block of colors to be repeated.
- The gradient repeats by shifting the color stops by the total length of the initial gradient segment.

- If colors differ at the transition points, the repetition will have a sharp visual cutoff, good for patterns like stripes.

Practical Use Cases

- Creating seamless striped backgrounds, grids, or textures.
- Repeating color bands for visual effects.

Understanding repeating linear gradients helps you create efficient, scalable patterned backgrounds without needing image files.

Radial Gradients -

Radial gradients in CSS create smooth color transitions radiating outward from a central point, often forming circles or ellipses. They differ from linear gradients, which transition colors along a straight line. Radial gradients start from a defined center and spread outward in circular or elliptical shapes.

Syntax

background-image: radial-gradient(shape size at position, start-color, ..., last-color);

Parameters Explained

- **shape** (optional): Defines the shape of the gradient. It can be:
  - circle — gradient forms a perfect circle
  - ellipse (default) — gradient forms an ellipse
- **size** (optional): Defines the size of the gradient. Common values:
  - closest-side — gradient ends at the closest side of the box from the center
  - farthest-side
  - closest-corner
  - farthest-corner (default) — gradient extends to the farthest corner
- **position** (optional): The origin point of the gradient. Default is center. You can specify positions like center, top left, or percentages like 40% 60%.
- **color stops**: At least two colors defining how the gradient transitions. You can optionally specify where the colors stop using percentages or lengths.

Basic Examples

1. **Simple radial gradient with default ellipse shape and center position:**

background-image: radial-gradient(red, yellow, green);

This creates a smooth gradient from red at the center to green at the edges, transitioning through yellow.

1. **Circular radial gradient:**

background-image: radial-gradient(circle, red, yellow, green);

Forces the gradient to be circular rather than elliptical.

1. **Radial gradient with position and size:**

background-image: radial-gradient(closest-side at 60% 55%, red, yellow, black);

Starts the gradient at point 60% from left and 55% from top; gradient ends at the closest side from that position.

1. **Radial gradient with differently spaced color stops:**

background-image: radial-gradient(red 5%, yellow 15%, green 60%);

Specifies exact positions for the colors, giving more control over the gradient transitions.

Repeating Radial Gradient

You can also create repeating radial gradients for patterns:

background-image: repeating-radial-gradient(red, yellow 10%, green 15%);

Repeats the gradient pattern continuously.

Practical Usage Tips

- Use radial gradients to create spotlight, ripple, or glow effects around elements.
- Combining multiple radial gradients can produce complex backgrounds and textures.
- Positioning the gradient's center allows for compositional control in your design.

Example in CSS

```
div {

  width: 300px;

  height: 300px;

  background-image: radial-gradient(circle farthest-corner at center, #ff0000, #ffff00, #00ff00);

}
```

This example creates a circular radial gradient starting at the center extending to the farthest corner, smoothly transitioning from red through yellow to green.

Understanding radial gradients enhances your ability to create visually appealing backgrounds and subtle design effects without images, improving performance and flexibility.

Box Shadow -

The CSS box-shadow property allows you to add shadow effects around an element's frame, giving a sense of depth and elevating the visual design of a webpage. It can create both outer shadows (default) and inner shadows (using the inset keyword).

Box Shadow Syntax

box-shadow: [horizontal-offset] [vertical-offset] [blur-radius] [spread-radius] [color] [inset];

- **Horizontal offset (required)**: Moves the shadow left (negative value) or right (positive value).
- **Vertical offset (required)**: Moves the shadow up (negative value) or down (positive value).
- **Blur radius (optional)**: Adds blur to the shadow; 0 means sharp edges. The higher the value, the softer the shadow.
- **Spread radius (optional)**: Expands (positive) or contracts (negative) the size of the shadow.
- **Color (optional)**: Defines the shadow color (can be color names, hex, rgba, hsla).
- **Inset (optional)**: Changes the shadow from an outer drop shadow to an inner shadow inside the element.

Examples

1. **Basic box shadow:**

```
div {
  box-shadow: 10px 10px 8px gray;
}
```

Creates a shadow 10px right and 10px down from the element, with 8px blur in gray color.

1. **Box shadow with spread radius:**

```
div {
  box-shadow: 10px 10px 5px 3px rgba(0, 0, 0, 0.5);
}
```

Adds a shadow that is shifted 10px right and down, blurred by 5px, and spread 3px larger, in semi-transparent black.

1. **Inner shadow:**

```
div {
  box-shadow: inset 5px 5px 10px rgba(0, 0, 0, 0.6);
}
```

Creates a shadow inside the element, shifted 5px right and down, blurred 10px, semi-transparent black.

1. **Multiple shadows:**

```
div {
  box-shadow: 3px 3px red, 6px 6px blue;
}
```

Creates two shadows: a red one shifted 3px right and down, and a blue one shifted 6px right and down.

Important Notes

- If you omit the color, the shadow uses the element's text color.
- Using semi-transparent colors (like rgba) for shadows is common to achieve natural shadow effects.
- Increasing blur radius makes shadows softer and more spread out.
- The inset keyword flips the shadow inside the element, useful for inner depth effects.
- You can combine multiple shadows separated by commas for layered effects.

Summary Table of Parameters

Parameter	Description	Example Value
Horizontal offset	Moves shadow left/right	10px (right), -5px (left)
Vertical offset	Moves shadow up/down	10px (down), -5px (up)
Blur radius	Softens shadow edges (0 = sharp)	8px, 0
Spread radius	Expands or contracts shadow size	3px, -2px
Color	Shadow color	gray, rgba(0,0,0,0.5)
Inset	Switches shadow from outer to inner	inset

This property is widely used to create depth, highlight elements, design "cards," buttons, modals, or floating UIs without images.

Drop Shadow -

The term "drop shadow" in CSS can refer to two related but distinct ways of creating shadow effects, distinguished mainly by how the shadow is applied and the effect it produces:

1. Box Shadow (box-shadow property)

- Applies a shadow around the rectangular box of an element, including its padding and border.
- The shadow forms a rectangular shape around the element's box, influenced by the element's dimensions and border radius (rounded corners).
- You can specify horizontal and vertical offsets, blur radius, spread radius, and color.
- Supports multiple shadows separated by commas.
- Efficient and widely supported.
- Example:
- css
- box-shadow: 10px 10px 8px rgba(0, 0, 0, 0.5);
- The shadow always follows the shape of the box (rectangle or rounded rectangle).

2. Drop Shadow (filter: drop-shadow() function)

- Applies a shadow that conforms precisely to the **visible shape** of the element, including transparent parts, such as non-rectangular or irregular shapes (e.g., images with transparency, SVGs).
- The shadow is based on the alpha channel of the element's rendered contents, so the shadow can perfectly match the shape.
- Takes offsets, blur, and color, but does **not** have a spread radius parameter.
- Supports one shadow per filter; multiple shadows require chaining multiple drop-shadow() filters.
- Can be more visually realistic for non-rectangular shapes but may be slower to render.
- Example:
- css
- filter: drop-shadow(5px 5px 10px rgba(0,0,0,0.5));
- Useful for applying shadows to images or elements with transparent backgrounds to keep natural contours.

Key Differences Between Box Shadow and Drop Shadow

Feature	Box Shadow	Drop Shadow
Applies shadow to	Rectangular box of element (according to box model)	Shape of element content including transparency
Affects	The element's frame (box shape)	Actual visible shape (alpha channel)
Supports multiple shadows	Yes, comma-separated	No, but can chain multiple filters
Spread radius	Supported	Not supported
Performance	Generally faster	Can be slower, GPU-accelerated in some cases
Use case	Most elements, UI components (cards, buttons)	Images, SVGs, irregular shapes

When to use which?

- Use **box-shadow** for general UI elements like buttons, cards, containers—anywhere the shadow should follow a rectangular shape.
- Use **drop-shadow** when you want the shadow to match the shape of complex images, logos, or elements with transparent backgrounds for a more natural effect.

Practical Example Comparison

```
/* Box Shadow example */

div.box-shadow {

  box-shadow: 10px 10px 8px rgba(0, 0, 0, 0.5);

}



/* Drop Shadow example */

img.drop-shadow {

  filter: drop-shadow(10px 10px 8px rgba(0, 0, 0, 0.5));

}
```

In this example, the div will have a rectangular shadow, while the image will have a shadow conforming to its visible, non-transparent shape.

Filters -

The CSS filter property allows you to apply various visual effects to elements such as images, backgrounds, text, or any other HTML elements. It manipulates how the browser renders the pixels of the element, modifying its appearance without changing the layout or the actual content.

Basic Syntax

```
selector {

  filter: filter-function(value);

}
```

You can also combine multiple filters by listing them separated by spaces:

```
filter: brightness(150%) contrast(120%);
```

Common CSS Filter Functions

- **blur(length):** Applies a Gaussian blur to the element. Example: blur(5px) blurs by 5 pixels.
- **brightness(% or number):** Adjusts the brightness. 100% is original brightness, less than 100% darkens, more than 100% brightens.
- **contrast(% or number):** Adjusts contrast. 100% is normal, less decreases contrast, more increases it.
- **drop-shadow(offset-x offset-y blur-radius color):** Applies a shadow that follows the shape of the element's visible content, not just the box. It accepts horizontal and vertical offsets (required), optional blur, and color.
- **grayscale(%):** Converts the element to grayscale. 0% is no effect, 100% is fully grayscale.
- **hue-rotate(angle):** Rotates the hue around the color circle. Example: hue-rotate(90deg).
- **invert(%):** Inverts the colors. 0% none, 100% fully inverted.
- **opacity(%):** Changes the transparency. 0% fully transparent, 100% opaque.
- **saturate(% or number):** Adjusts color saturation, with 100% being original.
- **sepia(%):** Applies a sepia tone, 0% none, 100% fully sepia.
- **url():** References an external SVG filter.

Example

```
img {

  filter: grayscale(100%) blur(3px) brightness(80%);

}
```

This example makes the image fully grayscale, slightly blurred, and less bright.

Notes

- The order of filters matters; for example, applying grayscale before or after brightness can produce different visual results.
- You can also reset filters with filter: none.
- Supported on most modern browsers.

Summary Table of Key Filters

Filter	Effect Description	Value Example
blur	Blurs the element by specified pixels	blur(5px)
brightness	Adjust brightness, as a percentage or number	brightness(150%)
contrast	Adjust contrast, percentage or number	contrast(120%)
drop-shadow	Adds shadow matching element shape	drop-shadow(10px 10px 5px black)
grayscale	Converts to grayscale	grayscale(100%)
hue-rotate	Rotates color hues around color wheel	hue-rotate(90deg)
invert	Inverts colors	invert(100%)
opacity	Adjust opacity	opacity(50%)
saturate	Adjust saturation	saturate(300%)
sepia	Applies sepia tone	sepia(100%)
url	Applies external SVG filter	url(filters.svg#filter-id)

Using the CSS filter property, you can create compelling visual effects efficiently and dynamically, enhancing UI design, image styling, hover feedback, and more.

List properties in css -

The main CSS properties for styling lists are related to the appearance and position of list item markers (like bullets or numbers) and can be summarized as follows:

1. list-style-type

- Specifies the **type of marker** used for list items.
- Common values for unordered lists: disc (default), circle, square, none.
- Common values for ordered lists: decimal (default, numbers), decimal-leading-zero, lower-roman, upper-roman, lower-alpha, upper-alpha, and more.
- Example:
- ul {
- list-style-type: circle;
- }
- ol {
- list-style-type: upper-roman;
- }

2. list-style-position

- Defines the **position of the list item marker** relative to the content.
- Values:
  - outside (default): Marker is outside the content flow (to the left).
  - inside: Marker is inside the content box, making the marker part of the text flow.
- Example:
- ul {
- list-style-position: inside;
- }

3. list-style-image

- Allows you to use a **custom image** as the list item marker instead of the default bullet or number.
- The value is a URL to an image.
- Example:
- ul {
- list-style-image: url('star.png');
- }

4. list-style (Shorthand property)

- A shorthand to set list-style-type, list-style-position, and list-style-image in one declaration.
- Syntax order can vary but usually: type, position, image.
- Example:
- ul {
- list-style: square inside url('icon.png');
- }

Summary Table

Property	Description	Example Values
list-style-type	Type of marker (bullet, number, etc.)	disc, circle, square, decimal, lower-alpha
list-style-position	Position of the marker relative to the text	outside, inside
list-style-image	Custom image as the marker	url('image.png')
list-style	Shorthand for type, position, and image	square inside url('icon.png')

Example Usage

```
ul.custom-list {
  list-style-type: square;
  list-style-position: inside;
  list-style-image: url('checkmark.png');
}

/* Using shorthand */
ol.custom-list {
  list-style: upper-roman outside url('star.png');
}
```

These properties allow you to fully customize the appearance of HTML lists for better visual presentation and usability.

Anchor state css -

The CSS anchor states are styled using **pseudo-classes** that represent the different states of a link (the <a> element). These pseudo-classes allow you to change the appearance of links depending on user interaction or link history.

Here are the four main anchor state pseudo-classes and their meanings:

1. **:link**
   Styles links that have not yet been visited by the user (unvisited links).
   Example: a:link { color: blue; }
2. **:visited**
   Styles links that the user has already visited (visited links).
   Example: a:visited { color: purple; }
3. **:hover**
   Styles links when the user's mouse pointer is over them (hover state).
   Example: a:hover { color: red; }
4. **:active**
   Styles links at the moment they are being clicked or activated (active state).
   Example: a:active { color: orange; }

Important Notes on CSS Order

The order in which you declare these pseudo-classes in your CSS rules matters for them to work correctly. The recommended order is:

```
a:link { /* unvisited link style */ }

a:visited { /* visited link style */ }

a:hover { /* hover style */ }

a:active { /* active style */ }
```

- The :hover rule must come **after** :link and :visited to override their styles when hovering.
- The :active rule must come **after** :hover to apply when the link is clicked.

Example CSS for Anchor States

```
a:link {

  color: blue;      /* Unvisited link */

  text-decoration: none;

}



a:visited {
```

```
  color: purple;    /* Visited link */
}

a:hover {
  color: red;       /* Mouse hover */
  text-decoration: underline;
}

a:active {
  color: orange;    /* Active (being clicked) link */
}
```

Summary Table

Pseudo-class	Description	Example Use
:link	Unvisited links	a:link { color: blue; }
:visited	Visited links	a:visited { color: purple; }
:hover	Link when mouse is over it	a:hover { color: red; }
:active	Link being clicked	a:active { color: orange; }

These pseudo-classes help create a more interactive and user-friendly website by visually indicating link states, improving navigation clarity.

Combinators CSS -

CSS combinators are special symbols or spacing used between selectors to define a relationship between those selectors in the HTML document structure. Combinators allow

you to select elements based on their position relative to other elements, making your CSS more powerful and specific.

There are **four main types of CSS combinators**:

1. Descendant Combinator (space)

- Represented by a **space** between selectors.
- Selects all elements that are descendants (children, grandchildren, etc.) of a specified element.
- Example:
- div p {
- color: blue;
- }
- This selects all <p> elements inside any , no matter how deeply nested.

2. Child Combinator (>)

- Represented by a **greater than sign (>).**
- Selects only the direct children of a specified element.
- Example:
- div > p {
- color: green;
- }
- This selects only <p> elements that are **immediate children** of a .

3. Adjacent Sibling Combinator (+)

- Represented by a **plus sign (+).**
- Selects an element that is the **next sibling** immediately following a specified element.
- Example:
- h2 + p {
- font-weight: bold;
- }
- This selects the first <p> that immediately follows an <h2> (both having the same parent).

4. General Sibling Combinator (~)

- Represented by a **tilde (~).**
- Selects all sibling elements that follow a specified element (not necessarily immediately).
- Example:
- h2 ~ p {
- color: red;
- }
- This selects all <p> siblings that come after any <h2> under the same parent.

Summary Table

Combinator	Symbol	What it Selects	Example	Description
Descendant Combinator	(space)	All descendants	div p	All <p> inside any .
Child Combinator	>	Direct children only	div > p	Immediate children <p> of .
Adjacent Sibling	+	Next sibling immediately after	h2 + p	<p> right after <h2>.
General Sibling	~	Any sibling following (not necessarily immediate)	h2 ~ p	All <p> siblings after <h2>.

Practical CSS Example

```
/* All <p> inside <div> */

div p {

  color: blue;

}


/* Only immediate <p> children of <div> */

div > p {

  color: green;

}


/* <p> immediately after <h2> */
```

```
h2 + p {
  font-weight: bold;
}

/* All <p> siblings following <h2> */
h2 ~ p {
  color: red;
}
```

Understanding and using CSS combinators will allow you to target elements very precisely based on their hierarchical or sibling relationships in the DOM, keeping your styles clean and efficient.

Display -

The CSS display property defines how an HTML element is displayed and how it participates in the layout of a webpage. It controls whether an element behaves like a block, inline element, flex container, grid container, or other layout types. This property is crucial because it determines the rendering behavior and positioning of elements and their children.

Key Display Property Values and Their Behavior

Value	Description	Example Use Case
block	Element takes full width available, starts on a new line; typical for container elements like .	, <p>, <section>
inline	Element flows inline with surrounding text, only takes up as much width as content requires.	, <a>,
inline-block	Behaves like inline, but allows setting width and height.	Buttons, image wrappers, tags
none	Element is not rendered and removed from the layout flow (invisible).	Hiding elements
flex	Makes the element a block-level flex container to arrange children using flexbox model.	Navigation menus, card layouts
inline-flex	Makes the element an inline-level flex container, behaving like inline but with flex layout.	Inline buttons with flexible children
grid	Makes the element a block-level grid container for two-dimensional layout of children.	Complex grid layouts
inline-grid	Makes the element an inline-level grid container.	Inline grid layouts
list-item	Element behaves like a list item (e.g.,), including	Custom list markers

	markers like bullets or	
table and related values (table-row, table-cell, etc.)	Makes element behave like a corresponding table part.	Styling elements as table components
contents	Makes the container disappear; the children behave as if they are direct children of the parent.	When you want to remove the wrapper but keep children in flow
inherit	Inherits the display value from its parent.	For consistent styling
initial	Resets to the default display value of the element as defined by the browser.	Resetting styles

Examples

```
/* Block element */
div {
  display: block;
}

/* Inline element */
span {
  display: inline;
}

/* Inline block - inline with block capabilities */
button {
  display: inline-block;
```

```
  width: 120px;
  height: 40px;
}

/* Flex container */
nav {
  display: flex;
  justify-content: space-between;
}

/* Grid container */
.section {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}

/* Hide element */
.hide {
  display: none;
}
```

Summary

- **Block elements** start on a new line and take the full width available.
- **Inline elements** flow within the line without disrupting text flow.
- **Inline-block** lets you combine inline flow with box-model controls (width, height).
- **Flex and grid** enable advanced layouts for children elements.
- **None** hides the element completely.
- Other values help simulate tables or lists or affect the display hierarchy via contents.

Understanding and using the display property effectively is fundamental to controlling layout and creating responsive, well-structured web pages.

Positions -

The CSS position property specifies how an element is positioned in the document and controls the element's layout behavior on a webpage. It determines whether the element is positioned according to the normal flow or taken out of it, and how you can manipulate its position with properties like top, right, bottom, and left.

There are **five main position values**:

1. static (default)

- Elements are positioned according to the normal flow of the document.
- The top, right, bottom, and left properties **do not** apply.
- No special positioning; elements appear where they naturally occur in the layout.
- Example use:
- div {
- position: static;
- }

2. relative

- Element is positioned **relative to its normal position** in the flow.
- You can move it using top, right, bottom, or left.
- Other content **does not adjust** to fill the space the element vacates.
- Useful for slight adjustments without removing the element from flow.
- Example:
- div {
- position: relative;
- top: 10px; */* moves down 10px from original position */*
- left: 20px; */* moves right 20px */*
- }

3. absolute

- Element is **removed from the normal flow** and positioned relative to the nearest positioned ancestor (ancestor with any position other than static).
- If no positioned ancestor exists, it positions relative to the initial containing block (usually the viewport).
- Uses top, right, bottom, and left to specify exact position.
- Other elements behave as if it is not there.
- Example:

- .container {
- position: relative;
- }
- div {
- position: absolute;
- top: 10px;
- left: 15px;
- }

4. fixed

- Positions the element relative to the **viewport**; stays fixed on the screen even when scrolling.
- Removed from normal flow.
- Uses top, right, bottom, and left for positioning within the viewport.
- Common for navigation bars, modals, or back-to-top buttons.
- Example:
- div {
- position: fixed;
- top: 0;
- right: 0;
- }

5. sticky

- Acts like relative until the element reaches a defined scroll position, then switches to fixed.
- Sticks to a specified offset (like top: 0) when scrolling past it.
- Useful for sticky headers or sidebars that scroll with the page initially but then fix in place.
- Example:
- div {
- position: sticky;
- top: 0;
- }

Summary Table

Position Value	Description	Effect on Layout Flow	Relation Reference
static	Default. Normal flow, unaffected by offset properties	Part of the normal document flow	Positioned as per the natural flow
relative	Moves relative to its normal position	Takes original space, content unaffected	Its own original spot
absolute	Removed from normal flow, positioned relative to nearest positioned ancestor	No space reserved for the element	Nearest positioned ancestor or viewport
fixed	Removed from flow, fixed in viewport	No space reserved, stays in viewport	Viewport (screen)
sticky	Switches between relative and fixed based on scroll position	Acts like relative until threshold is reached	Nearest scrolling ancestor or viewport

Practical Example Showing Different Positions

Static element

Relative element

Absolute element

Fixed element

Sticky element (try scrolling)

Understanding the position property is essential for layout control, stacking elements, and creating dynamic, interactive designs.

z-index -

The CSS z-index property controls the **stacking order** of elements that overlap on a webpage along the z-axis (depth). It determines which element appears **in front** or **behind** other elements when they overlap.

Key Points about z-index

- **Applies only to positioned elements**: Elements must have a position value other than static (e.g., relative, absolute, fixed, or sticky) or be flex/grid items.
- **Numeric values:** The property takes integer values (positive, zero, or negative).
  - Higher z-index values place elements **in front** of those with lower values.
  - Elements with **negative z-index** are placed behind those with zero or positive values.
- **Default behavior:** If no z-index is specified, elements stack in DOM order (later elements appear on top).
- **Special value**: auto which makes the element follow the stacking order of its parent context.

Syntax

z-index: auto | <integer> | initial | inherit;

How z-index works with positions

To use z-index, the element must be positioned:

```
.element {
  position: relative; /* or absolute, fixed, sticky */
  z-index: 10;
}
```

Example: Stacking Order

- The green div (z-index: 3) appears in front.
- The blue div (z-index: 2) comes next.
- The red div (z-index: 1) is at the back.

Negative z-index example

```
.background {

  position: absolute;

  z-index: -1;

}
```

The element will be behind other elements with default or positive z-index, often used for background layers.

Important Notes

- If two elements have the **same z-index**, their stacking order depends on their order in the HTML (the one later in the code is on top).
- z-index operates within **stacking contexts**. Some CSS properties create new stacking contexts (like opacity less than 1, transform, flex containers).
- If z-index isn't working, check the element has a proper position set.

Summary Table

Feature	Description
Applies to	Positioned elements (relative, absolute, fixed, sticky) and flex/grid items
Value Types	auto, integer (negative & positive), initial, inherit
Effect	Higher z-index values appear in front
Default stacking	DOM order when no z-index specified
Negative values	Element placed behind those with zero or positive values
Important usage tip	Must be positioned to work

Understanding z-index is essential for managing overlapping elements such as modals, dropdowns, tooltips, and layered UI components effectively.

Overflow -

The CSS overflow property controls what happens when an element's content is too large to fit within its designated box (width and/or height). It manages the behavior of content that spills outside the element's boundaries.

Key Values of overflow Property

Value	Description
visible (default)	Content is not clipped and overflows the element's box, remaining fully visible outside.
hidden	Overflowing content is clipped and **not visible**; no scrollbars are shown.
scroll	Content is clipped, but scrollbars are **always** added to allow viewing overflow content.
auto	Scrollbars are added **only when necessary**, if content overflows.
clip	Clips the overflow content but **forbids scrolling**, including programmatic scrolling.
initial	Resets to the default value (visible).
inherit	Inherits the overflow value from its parent element.

Additional Properties

- overflow-x: Controls overflow behavior **horizontally** (left and right edges).
- overflow-y: Controls overflow behavior **vertically** (top and bottom edges).

Practical Examples

```
/* Allow content to overflow and be visible */

div {

  overflow: visible; /* Default */

}



/* Hide content that overflows */
```

```
div {
  overflow: hidden;
}

/* Always show scrollbars */
div {
  overflow: scroll;
}

/* Show scrollbars only if needed */
div {
  overflow: auto;
}

/* Hide overflow but disallow scrolling */
div {
  overflow: clip;
}
```

Example using separate axes:

```
div {
  overflow-x: hidden; /* Hide horizontal overflow */
  overflow-y: scroll; /* Always show vertical scroll */
}
```

Important Notes

- The overflow property primarily affects **block elements with a specified height and/or width**.
- When overflow is visible, the content simply flows outside the element without clipping.

- Use hidden to clip overflowing content without scrollbars.
- Use scroll or auto to enable scrolling inside fixed-size containers.
- clip is stricter than hidden as it forbids any scrolling, even via scripts.
- Controlling overflow is crucial for maintaining clean layouts and user-friendly interfaces, especially in fixed-size containers or responsive designs.

Understanding and using the overflow property properly helps you control how excess content is handled, improving content visibility and UI behavior.

Pseudo Elements -

CSS pseudo-elements are special keywords added to selectors that allow you to style specific parts or aspects of an element without needing to add extra HTML markup. They effectively create "virtual" elements that let you target parts of an element's content or insert content before or after it.

Key Characteristics of Pseudo-elements:

- They target parts of an element, such as the first letter, first line, or content before/after the element.
- The syntax uses **double colons (::)** to distinguish them from pseudo-classes (which use a single colon :).
- Examples of pseudo-elements include ::before, ::after, ::first-letter, ::first-line, and more.
- Some pseudo-elements require a content property (like ::before and ::after) to insert generated content.

Commonly Used CSS Pseudo-elements

Pseudo-element	Purpose/Description	Example Usage
::before	Inserts content **before** the content of the element.	Add decorative icons or additional text.
::after	Inserts content **after** the content of the element.	Add clearfixes, decorations, or symbols.
::first-letter	Styles the **first letter** of the element's text.	Create drop caps or emphasize first letter.
::first-line	Styles the **first line** of the element's text.	Style first line differently (e.g., color).
::marker	Styles the **marker** (bullet or number) of list items.	Customize list bullets or numbers.
::selection	Styles the part of an element **selected** by the user.	Change highlight color when text is selected.
::placeholder	Styles the placeholder text in input and textarea elements.	Change placeholder font or color.

Examples

1. Add content before and after an element

```
p::before {
  content: "Note: ";
  font-weight: bold;
  color: red;
}

p::after {
```

```
  content: " ✓";
  color: green;
}
```

2. Style the first letter of a paragraph

```
p::first-letter {
  font-size: 3em;
  color: blue;
  float: left;
  margin-right: 5px;
}
```

3. Style the first line of text

```
p::first-line {
  font-weight: bold;
  color: darkorange;
}
```

4. Change the bullet color in unordered list items

```
li::marker {
  color: purple;
  font-size: 1.2em;
}
```

5. Customize how selected text looks

```
::selection {
  background: yellow;
  color: black;
}
```

Important Notes

- The double colon (::) notation is the current standard, but older browsers also support the single colon (:) for some pseudo-elements like :before and :after for backward compatibility.
- Pseudo-elements like ::before and ::after require the content property to render anything. Without content, they won't display.
- You cannot select child elements of a pseudo-element because they are not real elements in the DOM.
- Styling the ::selection pseudo-element can only apply a limited set of CSS properties such as color, background, cursor, and outline.

CSS pseudo-elements are powerful for adding styling details, inserting content dynamically, and creating sophisticated layouts without extra HTML markup, making your CSS both efficient and expressive.