Topics

1. CSS Introduction
2. CSS Syntax CSS Selectors CSS How To CSS Comments
3. CSS Colors
4. CSS Backgrounds
5. CSS Borders CSS Margins CSS Padding CSS Height/Width
6. CSS Box Model
7. CSS Outline CSS Text
8. CSS Fonts
9. CSS Icons
10. CSS Links
11. CSS Lists
12. CSS Tables CSS Display CSS Max-width
13. CSS Position CSS Z-index CSS Overflow CSS Float
14. CSS Inline-block
15. CSS Align
16. CSS Combinators CSS Pseudo-classes CSS Pseudo-elements CSS Opacity
17. CSS Navigation Bar CSS Dropdowns
18. CSS Image Gallery CSS Image Sprites CSS Attr Selectors CSS Forms
19. CSS Counters
20. CSS Website Layout CSS Units
21. CSS Specificity

# CSS Introduction

* **Description:** CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation of a document written in a markup language like HTML or XML (including various XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.
* **Key Concepts:**
* **Separation of Concerns:** CSS allows for the separation of document content (HTML) from its presentation (CSS), making websites easier to maintain and update.
* **Cascading:** Styles can originate from multiple sources (browser defaults, external stylesheets, internal stylesheets, inline styles). The "cascade" defines the order in which these styles are applied, with later declarations generally overriding earlier ones. Specificity and source order play crucial roles in the cascade.
* **Selectors:** CSS rules target specific HTML elements using selectors.
* **Properties and Values:** CSS rules consist of properties (e.g., color, font-size) and their corresponding values (e.g., red, 16px).
* **Use Cases:**
* Styling the visual appearance of web pages (colors, fonts, layout).
* Creating responsive designs that adapt to different screen sizes and devices.
* Implementing animations and transitions to enhance user experience.
* Applying consistent styling across multiple web pages.
* Generating print-friendly versions of web pages.

# CSS Syntax CSS Selectors CSS How To CSS Comments

* **CSS Syntax:**  
  selector {  
    property: value;  
    property: value;  
    ...  
  }
* **Selector:** Targets the HTML element(s) to be styled.
* **Declaration Block:** Enclosed in curly braces {}, contains one or more declarations.
* **Declaration:** Consists of a property and a value, separated by a colon : and ending with a semicolon ;.
* **CSS Selectors:** Define which HTML elements the CSS rules apply to.
* **Element Selector:** Selects elements based on their tag name (e.g., p, h1, div).  
  p { color: blue; }
* **ID Selector:** Selects a single element based on its unique id attribute (preceded by #).  
  <div id="main-content">...</div>  
    
  #main-content { background-color: yellow; }
* **Class Selector:** Selects elements with a specific class attribute (preceded by .). Multiple elements can have the same class.  
  <p class="important">...</p>  
  <div class="important">...</div>  
    
  .important { font-weight: bold; }
* **Attribute Selector:** Selects elements based on the presence or value of an attribute.
* [attribute]: Selects elements with the specified attribute.
* [attribute="value"]: Selects elements with the specified attribute and value.
* [attribute~="value"]: Selects elements with the specified attribute containing the value as a word.
* [attribute|="value"]: Selects elements with the specified attribute starting with the value followed by a hyphen or exactly equal to the value.
* [attribute^="value"]: Selects elements with the specified attribute value starting with the value.
* [attribute$="value"]: Selects elements with the specified attribute value ending with the value.
* [attribute\*="value"]: Selects elements with the specified attribute value containing the value.

<input type="text">  
<input type="password" data-theme="dark">  
input[type="text"] { border: 1px solid green; }  
input[data-theme="dark"] { background-color: black; color: white; }

* **Universal Selector:** Selects all HTML elements (\*).  
  \* { margin: 0; padding: 0; }
* **Grouping Selectors:** Apply the same styles to multiple selectors (separated by commas).  
  h1, h2, p { text-align: center; }
* **CSS How To:** Three ways to include CSS in HTML documents:
* **External Stylesheet:** Preferred method for larger projects. Link to a separate .css file using the <link> element in the <head> section of the HTML.  
  <head>  
    <link rel="stylesheet" href="styles.css">  
  </head>
* **Internal/Embedded Stylesheet:** Use the <style> element within the <head> section of the HTML. Suitable for single-page websites or specific styling.  
  <head>  
    <style>  
      h1 { color: purple; }  
    </style>  
  </head>
* **Inline Styles:** Apply styles directly to individual HTML elements using the style attribute. Should be used sparingly due to reduced maintainability.  
  <p style="color: red;">This is a red paragraph.</p>
* **CSS Comments:** Used to explain code and are ignored by the browser.  
  /\* This is a multi-line comment \*/  
  p {  
    color: green; /\* This is an inline comment \*/  
  }
* **Use Cases:**
* Targeting specific elements for individual styling.
* Applying consistent styles across groups of elements.
* Dynamically styling elements based on attributes.
* Organizing and managing styles effectively using external stylesheets.
* Adding notes and explanations to CSS code for better understanding and collaboration.

# CSS Colors

* **Description:** CSS provides various ways to specify colors for text, backgrounds, borders, and other properties.
* **Color Values:**
* **Named Colors:** Predefined color names (e.g., red, blue, green, black, white). A limited set of standard names is available.  
  h1 { color: navy; }
* **Hexadecimal (Hex) Codes:** Represent colors using a six-digit hexadecimal number (e.g., #FF0000 for red, #00FF00 for green, #0000FF for blue). Optionally, an eight-digit hex code can include an alpha (transparency) value (e.g., #FF000080 for 50% transparent red).  
  p { background-color: #f0f0f0; }
* **RGB (Red, Green, Blue):** Specifies color using the intensity of red, green, and blue components (values from 0 to 255).  
  div { border: 2px solid rgb(255, 165, 0); /\* Orange \*/ }
* **RGBA (Red, Green, Blue, Alpha):** Extends RGB by adding an alpha channel to specify transparency (value from 0.0 to 1.0).  
  span { color: rgba(0, 0, 255, 0.5); /\* Semi-transparent blue \*/ }
* **HSL (Hue, Saturation, Lightness):** Represents colors based on hue (color wheel degree), saturation (intensity), and lightness (brightness).  
  body { background-color: hsl(0, 100%, 50%); /\* Red \*/ }
* **HSLA (Hue, Saturation, Lightness, Alpha):** Extends HSL by adding an alpha channel for transparency.  
  .overlay { background-color: hsla(120, 60%, 70%, 0.8); /\* Semi-transparent light green \*/ }
* **Use Cases:**
* Setting text color for readability and visual hierarchy.
* Styling background colors for sections, containers, and the overall page.
* Defining border colors to visually separate elements.
* Creating subtle or vibrant color schemes to match branding.
* Implementing transparency and opacity effects.

# CSS Backgrounds

* **Description:** CSS background properties control the appearance of the background of HTML elements.
* **Background Properties:**
* background-color: Sets the background color of an element.  
  body { background-color: #e0f2f7; }
* background-image: Sets one or more background images for an element.  
  .hero { background-image: url('hero.jpg'); }
* background-repeat: Specifies how background images will be repeated (e.g., repeat, no-repeat, repeat-x, repeat-y).  
  .pattern { background-image: url('pattern.png'); background-repeat: repeat-x; }
* background-position: Specifies the initial position of the background image. Uses keywords (e.g., top, bottom, left, right, center) or pixel/percentage values.  
  .logo { background-image: url('logo.png'); background-position: top right; background-repeat: no-repeat; }
* background-size: Specifies the size of the background image. Can use keywords (auto, cover, contain) or pixel/percentage values.  
  .full-bg { background-image: url('large-image.jpg'); background-size: cover; }
* background-attachment: Sets whether the background image scrolls with the content (scroll) or is fixed in the viewport (fixed).  
  body { background-image: url('stars.gif'); background-attachment: fixed; }
* background-origin: Specifies the origin area for background images (padding-box, border-box, content-box).
* background-clip: Specifies the painting area of the background (border-box, padding-box, content-box, text).
* background: Shorthand property for setting multiple background properties in one declaration. The order of values matters.  
  .promo {  
    background: #f9f9f9 url('promo-bg.png') no-repeat center/cover fixed;  
  }
* **Use Cases:**
* Adding visual interest and branding to web pages.
* Creating patterns and textures.
* Implementing parallax scrolling effects.
* Optimizing image display using background-size.
* Styling specific sections with unique backgrounds.

# CSS Borders CSS Margins CSS Padding CSS Height/Width

* **CSS Borders:** Control the appearance of the border around an HTML element.
* border-width: Sets the thickness of the border (e.g., thin, medium, thick, pixel values).
* border-style: Sets the style of the border (e.g., none, solid, dashed, dotted, double, groove, ridge, inset, outset).
* border-color: Sets the color of the border.
* Individual border properties: border-top-width, border-right-style, border-bottom-color, etc.
* border-radius: Rounds the corners of an element's border.
* border: Shorthand property for setting border-width, border-style, and border-color in one declaration.  
  .box {  
    border: 1px solid black;  
    border-radius: 5px;  
  }
* **CSS Margins:** Define the space around the *outside* of an HTML element.
* margin-top, margin-right, margin-bottom, margin-left: Set margins for individual sides.
* margin: Shorthand property for setting all four margins.
* One value: applies to all four sides.
* Two values: top/bottom, left/right.
* Three values: top, left/right, bottom.
* Four values: top, right, bottom, left (clockwise).
* Auto margins (margin: 0 auto;) can horizontally center block-level elements within their parent.  
  .container {  
    margin: 20px;  
  }  
    
  .centered-block {  
    width: 50%;  
    margin: 20px auto;  
  }
* **CSS Padding:** Defines the space between the content of an HTML element and its border (the *inside* space).
* padding-top, padding-right, padding-bottom, padding-left: Set padding for individual sides.
* padding: Shorthand property similar to margin.  
  .content-area {  
    padding: 15px 30px; /\* top/bottom, left/right \*/  
  }
* **CSS Height/Width:** Set the height and width of an HTML element's content area.
* height: Specifies the height.
* width: Specifies the width.
* Values can be in pixels (px), percentages (%), ems (em), rems (rem), viewport units (vw, vh), etc.
* min-height, max-height, min-width, max-width: Set minimum and maximum constraints on the height and width.  
  .image-container {  
    width: 300px;  
    height: 200px;  
  }  
    
  img {  
    max-width: 100%; /\* Make images responsive within their container \*/  
    height: auto;  
  }
* **Use Cases:**
* Visually separating elements using borders.
* Creating spacing around elements to improve layout and readability (margins).
* Adding internal space within elements to prevent content from touching borders (padding).
* Controlling the size of elements for layout purposes.
* Creating responsive layouts using percentage-based widths and max-width.

# CSS Box Model

* **Description:** The CSS box model describes the rectangular boxes that are generated for HTML elements in the document tree and laid out according to visual formatting model. Each box consists of several parts:
* **Content:** The actual content of the element (text, images, etc.). Its dimensions are width and height.
* **Padding:** The space surrounding the content, inside the border.
* **Border:** A line that surrounds the padding and content.
* **Margin:** The space surrounding the border, outside the element.
* **Understanding the Box Model is Crucial for:**
* Accurately controlling the size and spacing of elements.
* Debugging layout issues.
* Creating consistent and predictable layouts.
* **box-sizing Property:** Alters the default box model behavior.
* content-box (default): width and height apply only to the content area. Padding and border are added to these dimensions, making the total size of the element larger than specified.
* border-box: width and height apply to the total size of the element, including padding and border. The content area shrinks to accommodate these. This model is often easier to work with for layout purposes.  
  .element {  
    width: 200px;  
    height: 100px;  
    padding: 20px;  
    border: 5px solid black;  
    /\* Total width with content-box: 200 + 20 + 20 + 5 + 5 = 250px \*/  
    /\* Total width with border-box: 200px (content shrinks to 150px) \*/  
    box-sizing: border-box;  
  }
* **Use Cases:**
* Predicting the actual rendered size of elements.
* Creating layouts where element widths and heights are easier to manage (using border-box).
* Understanding how padding and borders affect element dimensions.

# CSS Outline CSS Text

* **CSS Outline:** A line drawn *outside* the element's border. It does not affect the element's size or layout. Primarily used for accessibility to indicate focus.
* outline-width: Sets the thickness of the outline.
* outline-style: Sets the style of the outline (similar to border-style).
* outline-color: Sets the color of the outline.
* outline-offset: Specifies the space between the outline and the border.
* outline: Shorthand property.  
  button:focus {  
    outline: 2px solid blue;  
    outline-offset: 2px;  
  }
* **CSS Text:** Controls the appearance of text within an element.
* color: Sets the text color.
* direction: Sets the text direction (e.g., ltr - left-to-right, rtl - right-to-left).
* letter-spacing: Adjusts the space between characters.
* line-height: Sets the height of a line box.
* text-align: Specifies the horizontal alignment of text (left, right, center, justify).
* text-decoration: Adds or removes decorations from text (none, underline, overline, line-through). Can also specify color and style.
* text-indent: Indents the first line of text in a block-level element.
* text-shadow: Adds shadow effects to text.
* text-transform: Changes the case of text (uppercase, lowercase, capitalize).
* white-space: Controls how whitespace inside an element is handled (normal, nowrap, pre, pre-wrap, pre-line).
* word-spacing: Adjusts the space between words.
* word-break: Specifies how words should break if they are too long to fit (normal, break-all, keep-all).
* word-wrap (overflow-wrap): Allows long words to be broken and wrap onto the next line (normal, break-word).

```css p { color: #333; line-height: 1.6; text-align: justify; text-indent: 2em; }

h1 { text-align: center; text-transform: uppercase; letter-spacing: 0.1em; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3); }

a { text-decoration: none; } ```

* **Use Cases:**
* Enhancing text readability and visual appeal.
* Creating headings, paragraphs, and other text elements with specific styles.
* Implementing text-based effects like shadows and transformations.
* Controlling text flow and whitespace handling.
* Styling links and other inline text elements.
* Improving accessibility by providing visual focus indicators (outline).

# CSS Fonts

* **Description:** CSS font properties control the typeface, size, weight, and style of text.
* **Font Properties:**
* font-family: Specifies a list of font families to be used for an element. The browser will try to use the first available font in the list. It's good practice to include fallback fonts in case the preferred font is not available. Use generic font families (serif, sans-serif, monospace, cursive, fantasy) as fallbacks.  
  body {  
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;  
  }
* font-size: Sets the size of the font. Can use absolute units (px, pt) or relative units (em, rem, %).  
  h1 { font-size: 2.5em; }  
  p { font-size: 16px; }
* font-weight: Sets the boldness of the font (normal, bold, bolder, lighter, or numeric values from 100 to 900).  
  strong { font-weight: bold; }
* font-style: Sets the font style (normal, italic, oblique).  
  em { font-style: italic; }
* font-variant: Selects a variant of the font (e.g., normal, small-caps).  
  .small-caps { font-variant: small-caps; }
* @font-face: Rule used to embed custom fonts that are not installed on the user's computer.  
  @font-face {  
    font-family: 'MyCustomFont';  
    src: url('mycustomfont.woff2') format('woff2'),  
         url('mycustomfont.woff') format('woff');  
    font-weight: normal;  
    font-style: normal;  
  }  
    
  h1 {  
    font-family: 'MyCustomFont', sans-serif;  
  }
* font: Shorthand property for setting font-style, font-variant, font-weight, font-size, line-height, and font-family in one declaration.  
  p { font: italic bold 1.2em/1.5 "Open Sans", sans-serif; }
* **Use Cases:**
* Defining the visual style and readability of text.
* Implementing branding through specific font choices.
* Ensuring consistent typography across a website.
* Using web fonts to expand the available font options.
* Optimizing font loading for performance.

# CSS Icons

* **Description:** CSS can be used to incorporate icons into web pages in various ways.
* **Methods for Using Icons:**
* **Icon Fonts:** Fonts that contain glyphs of symbols and icons instead of letters and numbers (e.g., Font Awesome, Ionicons).
* **Pros:** Scalable vector graphics (SVG), easy to style with CSS (color, size), often come with a large library of icons.
* **Cons:** Can increase page load time if only a few icons are used from a large font file.
* **Implementation:** Link to the icon font stylesheet and use specific CSS classes provided by the font library in your HTML.

<i class="fas fa-home"></i>  
.fa-home {  
  font-size: 24px;  
  color: blue;  
}

* **SVG Icons:** Scalable Vector Graphics are XML-based image formats that can be styled with CSS.
* **Pros:** Scalable without loss of quality, can be styled with CSS (fill, stroke, etc.), can be embedded directly in HTML or linked as external files.
* **Cons:** More complex syntax than icon fonts.
* **Implementation:**
* **Inline SVG:** Embed the <svg> code directly in the HTML.
* **SVG as Image:** Use the <img> tag or background-image property with an SVG file. Limited CSS styling capabilities when used as <img>.
* **SVG as Object:** Use the <object> tag.
* **SVG Sprites:** Combine multiple SVG icons into a single file and use CSS to display specific icons.

<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="currentColor">  
  <path d="M12 2 L2 7 L12 12 L22 7 L12 2 Z M2 17 L12 22 L22 17 L12 12 L2 17 Z" />  
</svg>  
svg {  
  width: 24px;  
  height: 24px;  
  fill: green;  
}

* **CSS Shapes:** Using CSS properties like border-radius and clip-path to create simple icon-like shapes.
* **Pros:** Lightweight, no external files needed for basic shapes.
* **Cons:** Limited to simple geometric shapes.

.circle {  
  width: 30px;  
  height: 30px;  
  border-radius: 50%;  
  background-color: red;  
}

* **Image Sprites:** Combining multiple small images (icons) into a single image file. CSS background-position is used to display individual icons.
* **Pros:** Reduces HTTP requests.
* **Cons:** More complex to manage and update. Less flexible for styling individual icon colors.
* **Use Cases:**
* Enhancing user interface with visual cues.
* Providing visual representation of actions and statuses.
* Creating visually appealing navigation and controls.
* Improving website performance by using icon fonts or sprites.

# CSS Links

* **Description:** CSS can style the appearance of hyperlinks (<a> elements).
* **Link Pseudo-classes:** Used to style links based on their state:
* :link: Styles unvisited links.
* :visited: Styles links that have been visited.
* :hover: Styles links when the user hovers the mouse over them.
* :focus: Styles links when they have keyboard focus (e.g., after tabbing).
* :active: Styles links when they are being clicked (the moment the mouse button is pressed down).
* **Order matters:** :link and :visited should come before :hover, :hover should come before :active, and :focus can be placed anywhere but is often before :active. (LVHA - Love-Hate)
* **Common Link Properties:**
* color: Sets the text color of the link.
* text-decoration: Adds or removes text decorations (e.g., underline, none).
* background-color: Sets the background color of the link.
* font-weight: Changes the boldness of the link text.
* font-style: Changes the style of the link text (e.g., italic).
* cursor: Changes the mouse cursor when hovering over the link (e.g., pointer).  
  a:link {  
    color: blue;  
    text-decoration: none;  
  }  
    
  a:visited {  
    color: purple;  
    text-decoration: none;  
  }  
    
  a:hover {  
    color: red;  
    text-decoration: underline;  
  }  
    
  a:focus {  
    outline: 2px solid orange;  
  }  
    
  a:active {  
    color: green;  
  }
* **Use Cases:**
* Making links visually distinct from regular text.
* Providing feedback to the user on link interaction (hover, active, focus states).
* Maintaining consistent link styling across a website.
* Improving accessibility by providing clear visual focus indicators.

# CSS Lists

* **Description:** CSS can style unordered (<ul>) and ordered (<ol>) lists and their list items (<li>).
* **List Properties (applied to <ul> or <ol>):**
* list-style-type: Specifies the marker style for list items (e.g., disc, circle, square for <ul>; decimal, lower-alpha, upper-roman for <ol>; none to remove markers).
* list-style-position: Specifies the position of the list item markers (inside, outside).
* list-style-image: Allows using an image as the list item marker.
* list-style: Shorthand property for setting list-style-type, list-style-position, and list-style-image.
* **List Item Properties (applied to <li>):**
* Can be styled with standard text and box model properties.
* **Example:**  
  <ul>  
    <li>Coffee</li>  
    <li>Tea</li>  
    <li>Milk</li>  
  </ul>  
    
  <ol>  
    <li>First item</li>  
    <li>Second item</li>  
    <li>Third item</li>  
  </ol>  
    
  ul {  
    list-style-type: square;  
    padding-left: 20px;  
  }  
    
  ol {  
    list-style-type: upper-roman;  
  }  
    
  li {  
    margin-bottom: 5px;  
  }  
    
  .custom-marker {  
    list-style-image: url('checkmark.png');  
    list-style-position: inside;  
  }
* **Use Cases:**
* Presenting information in an organized and structured manner.
* Creating navigation menus (often by removing default list styles).
* Styling bullet points and numbered lists.
* Using custom images as list markers.

# CSS Tables CSS Display CSS Max-width

* **CSS Tables:** Styles the appearance of HTML tables (<table>, <tr>, <th>, <td>, etc.).
* **Border Properties:** border, border-collapse (collapses borders into a single border), border-spacing (sets space between table cell borders).
* **Padding:** padding for table cells (<td>, <th>).
* **Text Alignment:** text-align for table cells.
* **Vertical Alignment:** vertical-align for table cells (top, middle, bottom, etc.).
* **Background Color:** background-color for table elements.
* **Dimensions:** width, height for table and cells.
* **caption-side:** Positions the table caption (top, bottom).  
  <table>  
    <caption>Monthly Savings</caption>  
    <tr>  
      <th>Month</th>  
      <th>Savings</th>  
    </tr>  
    <tr>  
      <td>January</td>  
      <td>$100</td>  
    </tr>  
    <tr>  
      <td>February</td>  
      <td>$50</td>  
    </tr>  
  </table>  
    
  table {  
    width: 100%;  
    border-collapse: collapse;  
  }  
    
  th, td {  
    border: 1px solid #ddd;  
    padding: 8px;  
    text-align: left;  
  }  
    
  th {  
    background-color: #f2f2f2;  
  }  
    
  caption {  
    padding: 10px;  
    text-align: center;  
    font-weight: bold;  
  }
* **CSS Display:** Specifies the display type of an element, which affects how it behaves in the layout flow.
* block: Element takes up the full width available and starts on a new line (e.g., <div>, <p>, <h1>).
* inline: Element only takes up as much width as necessary and does not start a new line (e.g., <span>, <a>, <img>).
* inline-block: Element is like inline but allows setting width and height, as well as padding and margins on all sides.
* none: Element is not displayed at all (takes up no space).
* flex: Creates a flex container for flexible layout of its children.
* grid: Creates a grid container for two-dimensional layout.
* table, table-row, table-cell, etc.: For styling table-like structures (not necessarily HTML <table> elements).  
  .block-element { display: block; width: 50%; margin: 0 auto; }  
  .inline-element { display: inline; margin-right: 10px; }  
  .inline-block-element { display: inline-block; width: 100px; height: 50px; background-color: yellow; }  
  .hidden-element { display: none; }
* **CSS Max-width:** Sets the maximum width of an element. The actual width will be less than or equal to this value. Useful for creating responsive layouts where elements shouldn't become too wide on large screens.  
  .container {  
    max-width: 1200px;  
    margin: 0 auto; /\* Center the container \*/  
    padding: 20px;  
  }  
    
  img {  
    max-width: 100%; /\* Make images scale down within their container \*/  
    height: auto;  
  }
* **Use Cases:**
* Styling tables for data presentation.
* Controlling the layout behavior of elements (block, inline, inline-block).
* Hiding and showing elements dynamically (using display: none).
* Creating responsive layouts that adapt to different screen sizes using max-width.
* Building complex layouts with flex and grid (covered in more detail later).

# CSS Position CSS Z-index CSS Overflow CSS Float

* **CSS Position:** Specifies the positioning method used for an element.
* static (default): Elements are positioned according to the normal document flow. top, right, bottom, left, and z-index have no effect.
* relative: Element is positioned relative to its normal position in the document flow. Setting top, right, bottom, left offsets the element from its static position without affecting the layout of surrounding elements (creates a "hole" where it was).
* absolute: Element is removed from the normal document flow and positioned relative to its nearest *positioned* ancestor (an ancestor with a position value other than static). If no positioned ancestor exists, it's positioned relative to the initial containing block (the <html> element). Other elements will flow as if the absolutely positioned element doesn't exist.
* fixed: Element is removed from the normal document flow and positioned relative to the viewport. It stays in the same place even when the page is scrolled.

sticky: Element is positioned based on the normal flow, and then becomes fixed relative to the viewport when it reaches a specified offset (e.g., top: 0). Creates a "sticky" effect during scrolling.  
.relative { position: relative; top: 20px; left: 30px; }  
.absolute { position: absolute; top: 50px; right: 10px; } /\* Needs a positioned ancestor \*/  
.fixed { position: fixed; bottom: 0; left: 0; width: 100%; background-color: #f0f0f0; }  
.sticky { position: sticky; top: 0; background-color: white ; z-index: 10; } /\* For sticky navigation \*/ ```

* **CSS Z-index:** Specifies the stack order of positioned elements (elements with a position value other than static). An element with a higher z-index value will be in front of elements with lower z-index values. Can be positive, negative, or zero. Only works on positioned elements.  
  .overlay { position: absolute; top: 0; left: 0; background-color: rgba(0, 0, 0, 0.5); z-index: 1; }  
  .modal { position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); background-color: white; z-index: 10; } /\* Modal on top of overlay \*/
* **CSS Overflow:** Controls what happens when the content of an element is too big to fit into the specified area.
* visible (default): The overflow is not clipped, and the content may be displayed outside the element's box.
* hidden: The overflow is clipped, and the rest of the content will be invisible.
* scroll: The overflow is clipped, but a scrollbar is added to see the rest of the content (even if there's no overflow).
* auto: Similar to scroll, but scrollbars are only shown when the content overflows.
* overflow-x: Controls horizontal overflow.
* overflow-y: Controls vertical overflow.  
  .scrollable {  
    width: 200px;  
    height: 100px;  
    overflow: auto;  
    border: 1px solid #ccc;  
  }  
    
  .hidden-overflow {  
    width: 150px;  
    white-space: nowrap; /\* Prevent text wrapping \*/  
    overflow: hidden;  
    text-overflow: ellipsis; /\* Show ellipsis for hidden text \*/  
  }
* **CSS Float:** Positions an element to the left or right side of its container, allowing text and inline elements to wrap around it.
* left: Floats the element to the left.
* right: Floats the element to the right.
* none (default): Element is not floated.
* **Clearing Floats:** After floating elements, it's often necessary to "clear" the float to prevent subsequent elements from wrapping around it unexpectedly. Common methods:
* Adding an empty element with clear: both; (or clear: left; / clear: right;).

<div style="clear: both;"></div>

* Using the clearfix hack (applying a pseudo-element to the parent container):

.clearfix::after {  
  content: "";  
  display: table;  
  clear: both;  
}  
  
.container {  
  overflow: auto; /\* Another way to clear floats \*/  
}

* **Use Cases:**
* Creating complex layouts with overlapping elements (position, z-index).
* Implementing scrolling containers for content that exceeds its boundaries (overflow).
* Creating layouts where text wraps around images or other floated elements (float).
* Building navigation menus and multi-column layouts using floats (though Flexbox and Grid are often preferred for modern layouts).
* Creating sticky headers or footers (position: sticky, position: fixed).

# CSS Inline-block

* **Description:** The display: inline-block; property is a hybrid of inline and block.
* **Characteristics:**
* Like inline elements, they only take up as much width as necessary and do not force line breaks before or after them.
* Like block elements, you can set their width and height, as well as padding and margin on all four sides (inline elements only respect horizontal padding and margin).
* **Use Cases:**
* Creating horizontal navigation menus where list items are displayed in a row but can be styled with width, height, and padding.
* Arranging elements side-by-side without using floats.
* Giving inline elements block-level styling capabilities.

<div class="toolbar">  
  <button style="display: inline-block; padding: 10px 20px; margin-right: 10px;">Button 1</button>  
  <button style="display: inline-block; padding: 10px 20px;">Button 2</button>  
</div>

# CSS Align

* **Description:** CSS provides various properties for aligning elements and their content. The specific properties used depend on the layout context (e.g., block layout, inline layout, Flexbox, Grid).
* **Common Alignment Properties:**
* **text-align:** Aligns text content within a block-level element (left, right, center, justify).
* **vertical-align:** Aligns inline, inline-block, and table-cell content vertically with respect to their containing line box or table row (top, middle, bottom, baseline, etc.).
* **margin: 0 auto;:** Horizontally centers block-level elements within their parent container (by setting equal left and right auto margins).
* **Flexbox Alignment:** Properties applied to the flex container (display: flex;) and flex items (children of the flex container):
* justify-content: Aligns flex items along the main axis (flex-start, flex-end, center, space-between, space-around, space-evenly).
* align-items: Aligns flex items along the cross axis (flex-start, flex-end, center, baseline, stretch).
* align-self: Aligns a specific flex item along the cross axis, overriding the align-items value.
* **Grid Layout Alignment:** Properties applied to the grid container (display: grid;) and grid items (children of the grid container):
* justify-items: Aligns grid items along the inline (row) axis within their grid cells (start, end, center, stretch).
* align-items: Aligns grid items along the block (column) axis within their grid cells (start, end, center, stretch).
* justify-self: Aligns a specific grid item along the inline axis within its grid cell.
* align-self: Aligns a specific grid item along the block axis within its grid cell.
* place-items: Shorthand for align-items and justify-items.
* place-self: Shorthand for align-self and justify-self.
* justify-content: Aligns the grid tracks along the inline axis within the grid container.
* align-content: Aligns the grid tracks along the block axis within the grid container (start, end, center, space-between, space-around, space-evenly, stretch).
* place-content: Shorthand for align-content and justify-content.
* **Use Cases:**
* Centering text within buttons or containers.
* Vertically aligning icons with text.
* Creating flexible and responsive layouts using Flexbox and Grid where alignment of items is crucial.
* Distributing space between elements.

# CSS Combinators CSS Pseudo-classes CSS Pseudo-elements CSS Opacity

* **CSS Combinators:** Define the relationship between selectors.
* **Descendant Selector (space):** Selects all elements that are descendants of a specified element (e.g., div p selects all <p> elements inside <div>).
* **Child Selector (>):** Selects only elements that are direct children of a specified element (e.g., ul > li selects only <li> elements that are direct children of <ul>).
* **Adjacent Sibling Selector (+):** Selects the element that is the immediately following sibling of a specified element (e.g., h2 + p selects the first <p> element that immediately follows an <h2>).
* **General Sibling Selector (~):** Selects all sibling elements that follow a specified element (e.g., h2 ~ p selects all <p> elements that are siblings of an <h2> and come after it).
* **CSS Pseudo-classes:** Select elements based on their state or relation to the document tree, rather than their name, attributes, or content. Start with a colon :.
* **User Action Pseudo-classes:** :hover, :active, :focus, :visited, :link.
* **Structural Pseudo-classes:** :first-child, :last-child, :nth-child(n), :nth-last-child(n), :first-of-type, :last-of-type, :nth-of-type(n), :nth-last-of-type(n), :only-child, :only-of-type, :empty.
* **Form Pseudo-classes:** :enabled, :disabled, :checked, :required, :optional, :valid, :invalid, :in-range, :out-of-range, etc.
* **Language Pseudo-class:** :lang(language).
* **CSS Pseudo-elements:** Allow you to style specific parts of an element. Start with a double colon ::.
* ::before: Inserts generated content before the content of an element.
* ::after: Inserts generated content after the content of an element. Often used with the content property.
* ::first-line: Styles the first line of a block-level element.
* ::first-letter: Styles the first letter of a block-level element.
* ::selection: Styles the portion of an element that is selected by the user.
* ::placeholder: Styles the placeholder text of a form input element.
* **CSS Opacity:** Specifies the transparency of an element. The value ranges from 0.0 (fully transparent) to 1.0 (fully opaque). Affects the entire element, including its content.  
  div p { color: green; } /\* Descendant \*/  
  ul > li { border-left: 5px solid blue; } /\* Child \*/  
  h2 + p { font-weight: bold; } /\* Adjacent sibling \*/  
  h2 ~ p { text-decoration: underline; } /\* General sibling \*/  
  button:hover { background-color: #eee; } /\* Pseudo-class \*/  
  li:nth-child(even) { background-color: #f9f9f9; } /\* Pseudo-class \*/  
  .notification::before { content: "!"; color: red; font-weight: bold; margin-right: 5px; } /\* Pseudo-element \*/  
  .overlay { opacity: 0.8; } /\* Opacity \*/
* **Use Cases:**
* Targeting specific elements based on their position in the DOM or their state.
* Adding decorative elements or content using ::before and ::after.
* Styling specific parts of text or form elements.
* Creating visual effects with transparency.

# CSS Navigation Bar CSS Dropdowns

* **CSS Navigation Bar:** Styles the appearance and layout of navigation menus. Common techniques involve using unordered lists (<ul>) and styling the list items (<li>) and links (<a>).
* **Horizontal Navigation:** Use display: inline or display: inline-block on list items to arrange them horizontally. Remove list item markers with list-style-type: none;. Style links for visual appeal and hover effects.
* **Vertical Navigation:** Default list item behavior (block-level) often works for vertical navigation. Style links and add hover effects.
* **Fixed Navigation:** Use position: fixed; to keep the navigation bar at a specific location on the screen during scrolling.

<nav>  
  <ul>  
    <li><a href="#">Home</a></li>  
    <li><a href="#">About</a></li>  
    <li><a href="#">Services</a></li>  
    <li><a href="#">Contact</a></li>  
  </ul>  
</nav>  
nav ul {  
  list-style-type: none;  
  margin: 0;  
  padding: 0;  
  background-color: #333;  
  overflow: hidden; /\* Clear floats if needed \*/  
}  
  
nav ul li {  
  float: left; /\* For horizontal navigation \*/  
}  
  
nav ul li a {  
  display: block;  
  color: white;  
  text-align: center;  
  padding: 14px 16px;  
  text-decoration: none;  
}  
  
nav ul li a:hover {  
  background-color: #111;  
}

* **CSS Dropdowns:** Create menus that appear when a user hovers over or clicks on a trigger element. Typically involve nested lists and CSS to control visibility.
* **Basic Dropdown (Hover-based):**

1. Create a main list item with a submenu (another <ul>).
2. Initially hide the submenu using display: none;.
3. Use the :hover pseudo-class on the main list item to change the display property of the submenu to block.
4. Style the submenu as needed.

* **Advanced Dropdowns:** May involve more complex CSS for positioning, animations, and keyboard navigation.

<div class="dropdown">  
  <button class="dropbtn">Menu</button>  
  <div class="dropdown-content">  
    <a href="#">Link 1</a>  
    <a href="#">Link 2</a>  
    <a href="#">Link 3</a>  
  </div>  
</div>  
.dropdown {  
  position: relative;  
  display: inline-block;  
}  
  
.dropbtn {  
  background-color: #4CAF50;  
  color: white;  
  padding: 16px;  
  font-size: 16px;  
  border: none;  
  cursor: pointer;  
}  
  
.dropdown-content {  
  display: none;  
  position: absolute;  
  background-color: #f9f9f9;  
  min-width: 160px;  
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);  
  z-index: 1;  
}  
  
.dropdown-content a {  
  color: black;  
  padding: 12px 16px;  
  text-decoration: none;  
  display: block;  
}  
  
.dropdown-content a:hover {background-color: #ddd;}  
  
.dropdown:hover .dropdown-content {display: block;}

* **Use Cases:**
* Providing website navigation.
* Organizing large numbers of links.
* Creating interactive menu systems.

# CSS Image Gallery CSS Image Sprites CSS Attr Selectors

* **CSS Image Gallery:** Styles a collection of images, often with layouts like grids or carousels.
* **Grid Layout:** Use CSS Grid or Flexbox to arrange images in rows and columns.
* **Thumbnails:** Display smaller versions of images that link to larger versions or open in a lightbox.
* **Captions:** Add descriptive text below or overlaid on images.
* **Hover Effects:** Implement visual changes on image hover (e.g., opacity, zoom, border).

<div class="gallery">  
  <img src="image1.jpg" alt="Image 1">  
  <img src="image2.jpg" alt="Image 2">  
  <img src="image3.jpg" alt="Image 3">  
</div>  
.gallery {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));  
  gap: 20px;  
  padding: 20px;  
}  
  
.gallery img {  
  width: 100%;  
  height: auto;  
  border-radius: 5px;  
  box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3);  
  transition: transform 0.3s ease-in-out;  
}  
  
.gallery img:hover {  
  transform: scale(1.05);  
}

* **CSS Image Sprites:** Combine multiple small images into a single image file. Use the background-image and background-position properties to display individual images as needed.
* **Benefits:** Reduces the number of HTTP requests, potentially improving page load time.
* **Implementation:** Create the sprite sheet, then for each icon, set the background-image to the sprite sheet and adjust the background-position to show the desired icon. You'll also need to set the width and height of the element to the dimensions of the individual icon.

<div class="icon home-icon"></div>  
<div class="icon search-icon"></div>  
.icon {  
  width: 32px;  
  height: 32px;  
  background-image: url('icons-sprite.png');  
  background-repeat: no-repeat;  
}  
  
.home-icon {  
  background-position: 0 0; /\* Top-left corner of the sprite \*/  
}  
  
.search-icon {  
  background-position: -32px 0; /\* Next icon in the sprite \*/  
}

* **CSS Attribute Selectors:** Select HTML elements based on the presence or value of their attributes (covered previously in section 2). \* **Use Cases:** Styling elements with specific attributes (e.g., all links with target="\_blank"), applying different styles based on data attributes, styling form elements based on their type. ```css a[target="\_blank"] { color: orange; }

input[type="text"] { border: 1px solid #ccc; padding: 8px; }

button[disabled] { opacity: 0.5; cursor: not-allowed; } ```

# CSS Forms

* Styles HTML form elements (<input>, <textarea>, <select>, <button>, etc.).
* **Basic Styling:** Use standard CSS properties like color, background-color, border, padding, margin, font-size.
* **Pseudo-classes:** Style form elements based on their state (e.g., :focus, :valid, :invalid, :required, :disabled).
* **Appearance Property:** Controls the native appearance of form controls (none to remove native styling).
* **Placeholder Styling:** Use the ::placeholder pseudo-element to style placeholder text.
* **Custom Checkboxes and Radio Buttons:** Often involve hiding the default elements and using labels with custom styling (e.g., using ::before or background-image).

<form>  
  <div>  
    <label for="name">Name:</label>  
    <input type="text" id="name" required placeholder="Your name">  
  </div>  
  <div>  
    <label for="email">Email:</label>  
    <input type="email" id="email">  
  </div>  
  <button type="submit">Submit</button>  
</form>  
input[type="text"],  
input[type="email"] {  
  width: 100%;  
  padding: 10px;  
  margin-bottom: 10px;  
  border: 1px solid #ddd;  
  box-sizing: border-box;  
}  
  
input:focus {  
  outline: none;  
  border-color: blue;  
  box-shadow: 0 0 5px rgba(0, 0, 255, 0.3);  
}  
  
input::placeholder {  
  color: #aaa;  
}  
  
button[type="submit"] {  
  background-color: #4CAF50;  
  color: white;  
  padding: 12px 20px;  
  border: none;  
  cursor: pointer;  
}  
  
input:invalid {  
  border-color: red;  
}

* **Use Cases:**
* Creating visually appealing and user-friendly image galleries.
* Optimizing website performance with image sprites.
* Targeting elements based on their attributes for specific styling.
* Styling form elements to match the website's design and provide clear visual feedback to users.

# CSS Counters

* **Description:** CSS Counters allow you to number elements automatically using CSS. Useful for creating outlines, numbered lists with custom styling, and more.
* **Counter Properties:**
* counter-reset: Creates or resets a counter. Specified on the parent element.
* counter-reset: my-counter; (resets my-counter to 0)
* counter-reset: chapter section 1; (resets chapter to 0, section to 1)
* counter-increment: Increments the value of a counter. Specified on the element to be numbered.
* counter-increment: my-counter; (increments my-counter by 1)
* counter-increment: section 2; (increments section by 2)
* counter(): Used with the content property in a ::before or ::after pseudo-element to display the current value of a counter.
* content: counter(my-counter);
* content: counter(my-counter, upper-roman); (formats the counter value)
* counters(): Similar to counter(), but can display nested counters with a specified separator (e.g., for section numbering like 1.1, 1.2, 2.1).
* content: counters(chapter, ".") "." counter(section);
* **Example (Numbered Headings):**  
  <h1>Chapter 1</h1>  
  <h2>Section 1.1</h2>  
  <h2>Section 1.2</h2>  
  <h1>Chapter 2</h1>  
  <h2>Section 2.1</h2>  
    
  body {  
    counter-reset: chapter; /\* Reset chapter counter on the body \*/  
  }  
    
  h1 {  
    counter-increment: chapter; /\* Increment chapter counter for each h1 \*/  
    counter-reset: section; /\* Reset section counter for each new chapter \*/  
  }  
    
  h1::before {  
    content: "Chapter " counter(chapter) ": ";  
  }  
    
  h2 {  
    counter-increment: section; /\* Increment section counter for each h2 \*/  
  }  
    
  h2::before {  
    content: counters(chapter, ".") "." counter(section) " ";  
  }
* **Use Cases:**
* Automatically numbering headings, sections, or list items with custom formatting.
* Creating dynamic outlines.
* Implementing complex numbering schemes.

# CSS Website Layout CSS Units

* **CSS Website Layout:** Involves structuring and arranging the different parts of a web page. Modern CSS offers several powerful layout techniques:
* **Normal Flow:** The default way elements are laid out (block elements stack vertically, inline elements flow horizontally).
* **Floats:** (Covered in section 13) Used to create wrapping text around elements, but can lead to layout complexities.
* **Flexbox (Flexible Box Layout):** A one-dimensional layout system for arranging items in rows or columns. Excellent for aligning and distributing space among items within a container.
* **Key Properties (Container):** display: flex, flex-direction, justify-content, align-items, flex-wrap, align-content.
* **Key Properties (Items):** order, flex-grow, flex-shrink, flex-basis, align-self.
* **Grid Layout:** A two-dimensional layout system for creating complex grid-based structures. Allows precise control over the placement and sizing of elements in rows and columns.
* **Key Properties (Container):** display: grid, grid-template-rows, grid-template-columns, grid-template-areas, grid-row-gap, grid-column-gap, gap, justify-items, align-items, justify-content, align-content.
* **Key Properties (Items):** grid-row-start, grid-row-end, grid-column-start, grid-column-end, grid-area, justify-self, align-self.
* **Multi-column Layout:** CSS Columns allow you to create newspaper-like layouts with text flowing across multiple columns (column-count, column-width, column-gap, column-rule).
* **CSS Units:** Specify the size and length values in CSS.
* **Absolute Units:** Fixed sizes that are generally the same regardless of the surrounding elements or screen size (e.g., px (pixels), pt (points), cm (centimeters), mm (millimeters), in (inches)).
* **Relative Units:** Sizes that are relative to another value, making layouts more flexible and scalable.
* **Font-relative:**
* em: Relative to the font-size of the current element.
* rem: Relative to the font-size of the root element (<html>).
* ex: Relative to the x-height of the font.
* ch: Relative to the width of the "0" (zero) character of the font.
* **Viewport-percentage lengths:**
* vw: 1% of the viewport width.
* vh: 1% of the viewport height.
* vmin: 1% of the smaller side of the viewport.
* vmax: 1% of the larger side of the viewport.
* **Other relative units:**
* %: Percentage of the parent element's property value.
* **Choosing Units:**
* px: Good for fine-tuning borders and for situations where a fixed size is required.
* em and rem: Excellent for creating scalable and maintainable designs based on typography. rem is often preferred for global font sizing.
* %: Useful for creating responsive layouts based on parent element dimensions.
* vw and vh: Ideal for full-viewport elements and creating layouts that adapt to the screen size.
* **Use Cases:**
* Structuring the overall layout of web pages (header, navigation, main content, sidebar, footer).
* Creating responsive designs that adapt to different screen sizes and devices using Flexbox, Grid, and relative units.
* Implementing complex and flexible component layouts.
* Designing print-friendly layouts using appropriate units.

# CSS Specificity

* **Description:** Specificity is the algorithm that browsers use to determine which CSS rule applies to an element when multiple conflicting rules are declared. Rules with higher specificity override rules with lower specificity.
* **Specificity Hierarchy (from highest to lowest):**

1. **Inline styles:** Styles applied directly to an HTML element using the style attribute.
2. **IDs:** Selectors that target an element by its id attribute (#myid).
3. **Classes, pseudo-classes, and attributes:** Selectors that target elements by their class attribute (.myclass), pseudo-classes (:hover), or attributes ([type="text"]).
4. **Elements and pseudo-elements:** Selectors that target elements by their tag name (div) or pseudo-elements (::before).
5. **Universal selector (\*), combinators ( , >, +, ~), and the inherit value:** These have a specificity of 0.

* **Calculating Specificity:** Think of specificity as a four-column value (inline, IDs, classes/attributes/pseudo-classes, elements/pseudo-elements).
* Inline styles get 1 in the first column.
* Each ID selector adds 1 to the second column.
* Each class selector, attribute selector, and pseudo-class adds 1 to the third column.
* Each element selector and pseudo-element adds 1 to the fourth column.
* Combinators and the universal selector do not add to the specificity value.
* **Example:**  
  /\* 1 element: specificity 0,0,0,1 \*/  
  div { color: blue; }  
    
  /\* 1 class: specificity 0,0,1,0 \*/  
  .container { color: green; }  
    
  /\* 1 ID: specificity 0,1,0,0 \*/  
  #main-title { color: red; }  
    
  /\* Inline style: specificity 1,0,0,0 (highest) \*/  
  <p style="color: purple;">This text will be purple.</p>  
    
  /\* 1 element + 1 class: specificity 0,0,1,1 \*/  
  p.container { color: orange; } /\* Overrides just 'p' \*/  
    
  /\* 1 ID + 1 class: specificity 0,1,1,0 \*/  
  #main-title.large { color: yellow; } /\* Overrides just '#main-title' \*/
* **!important Rule:** Can be used to override specificity, but should be used sparingly as it can make debugging difficult. It should generally be reserved for utility classes or critical overrides.  
  p { color: blue !important; } /\* This style will likely override other p styles \*/
* **Use Cases:**
* Understanding why certain CSS rules are being applied and others are being overridden.
* Writing more targeted CSS selectors to ensure the correct styles are applied.
* Debugging CSS issues related to conflicting styles.
* Using specificity effectively to manage and organize CSS rules in larger projects.