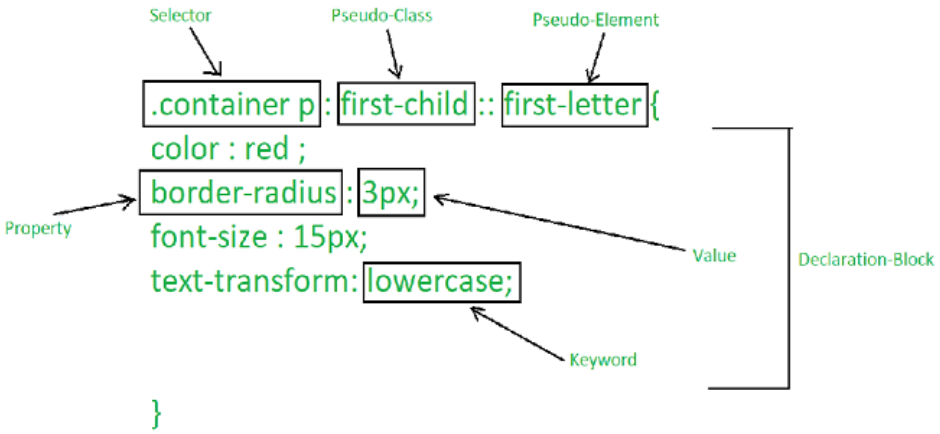
CSS 100+ Interview Q&A

By CodeBustler

1. **What do you understand about the universal * selector ?**
   The universal * selector in CSS **matches and applies styles to all elements** on a webpage. It's a way to target everything at once.

2. **What is Cascading?**
   Cascading in CSS **refers to how styles from various sources combine** and interact to determine the final appearance of elements. This process involves prioritizing styles based on specificity, inheritance, and order of declaration

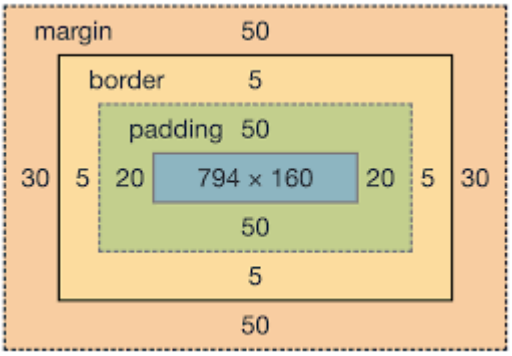
3. **What is the ruleset in CSS ?**
   A ruleset in CSS consists of a **selector** and a **set of declarations**. The selector determines which HTML elements the rule applies to, and the declarations specify the styling properties and values for those elements. It defines **how elements should be styled** on a webpage



4. **What is the CSS Box Model & its elements ?**

   The CSS Box Model is a concept that defines how elements on a webpage are **structured** in terms of size and spacing. It consists o**f four main elements**:

   Content : The actual element's content, like text or images.
   Padding : Space between the content and the border.
   Border : The element's boundary, surrounding the padding and content.
   Margin : Space outside the element, providing separation from other elements.



5. **What is CSS ? Explain with its versions**
   CSS (**Cascading Style Sheets**) is a language **used to style** and format HTML documents, controlling their **appearance on the web**. It separates the visual design from the document's structure.

   <u>There are different versions:</u>
   CSS1 : Introduced basic styling properties.
   CSS2 : Added more advanced features like positioning and typography.
   CSS2.1 : A revision that addressed issues in CSS2.
   CSS3 : Modular update with various modules for advanced features like **transitions**, **flexbox**, **grid** and more.
   CSS4 : Not an official version but represents ongoing module-based updates for advanced styling capabilities.

6. **Difference between CSS3 & CSS2 ?**
   CSS3 is an upgraded version of CSS2 that brings new and advanced styling capabilities to web design. It introduces modules for
   - animations,
   - transitions,
   - flexible layouts (Flexbox),
   - grid layouts (Grid), and more.

   It offers improved typography, rounded corners, shadows, and better control over styling effects. CSS3 allows for responsive design through media queries. Its modular approach and expanded features make it more versatile and powerful compared to CSS2.

7. **What do you mean by RGB stream ?**
   An "RGB stream" refers to a continuous flow of color data in the form of **Red, Green, and Blue** components. Each parameter (red, green, and blue) defines the intensity of the color with a value between **0 and 255**.
   rgb(red, green, blue) | rgb(255, 255, 255) = Generate white color
   rgba(red, green, blue, alpha) alpha for transparent

8. **What was the purpose of developing CSS?**
   The purpose of developing CSS (Cascading Style Sheets) was to **separate the presentation or styling of web documents** from their underlying structure. Prior to the introduction of CSS, web designers had to include styling directly within HTML

documents, which made it difficult to maintain and modify the design consistently across multiple pages.

9. **Name some CSS Frameworks**
    - Bootstrap
    - Foundation
    - Materialize
    - Bulma
    - Semantic UI
    - Tailwind CSS
    - UIKit
    - Pure.css
    - Material-UI

10. **Difference between a " class " & an "Id" ?**
    Class: Used to group multiple elements together for applying the same styling. Can be applied to multiple elements, and you target it in CSS using a dot **(.)**

    ID: Used to uniquely identify a single element. Each ID must be unique in the document. You target it in CSS using a hash **(#)**.

11. **What are CSS Sprites ? What are the benefits**
    CSS sprites involve **combining multiple images into one** and using CSS background positioning to display specific parts of the combined image.

    Fewer Requests : Reduced HTTP requests.
    Faster Loading : Faster page loading times.
    Better Performance : Improved user experience.
    Efficient Caching : Browser caches a single image.
    Simplified CSS : Manage image positions in CSS.
    Reduced Latency : Minimized request delays.
    Mobile Optimization : Improved mobile performance.
    Responsive Friendly: Supports responsive designs.

```css
#sprite{
    height: 128px;
    background-image: url('images/sprite.png');
    background-repeat: no-repeat;
    background-position: 0px -288px;
}
```

12. **How can you use CSS to control image repetition**
    CSS's **background-repeat** property controls image repetition in an element's background:

    repeat: Default, image repeats both horizontally and vertically.
    repeat-x: Image repeats only horizontally.

```css
.container p {
    color: blue;
}
```

repeat-y: Image repeats only vertically.
no-repeat : Image is not repeated.

13. **Define contextual selectors**
    Contextual selectors in CSS target elements based on their **hierarchical relationship** within the HTML structure. They are written as **parentElement descendantElement** and apply styles to the descendant elements that are within the specified parent element.

14. **Explain responsive web design**
    Responsive web design (RWD) is an approach to web design & development that aims to create websites that **automatically adapt and provide an optimal viewing experience on various devices** & screen sizes, including desktops, laptops, tablets, and smartphones.

    Key principles of responsive web design
    - Fluid Grids
    - Flexible Images
    - Media Queries
    - Viewport Meta Tag
    - Content Prioritization
    - Breakpoints

15. **How to optimize the website for prints ?**

    - **Create Print Stylesheet**: Make a separate CSS stylesheet for printing. **@media print { ... }.**
    - **Remove Unnecessary Elements**: Hide irrelevant items like menus.
    - **Adjust Fonts and Colors:** Opt for legible fonts and colors.
    - **Remove Background Images:** Avoid printing background images.
    - **Control Page Breaks:** Use CSS to prevent awkward content splits.
    - **Increase Margins and Padding:** Add white space for readability.
    - **Test Printing**: Print and adjust to ensure content fits well.
    - **Offer Printer-Friendly Format**: Provide simplified content version.
    - **PDF Option:** Offer downloadable PDFs for printing.
    - **Headers and Footers:** Use CSS for page info.
    - **Check Images:** Ensure graphics are print-friendly

16. **What is CSS “working under the hood” ?**
    CSS applies styles to HTML elements, controlling their appearance. Browsers load HTML and CSS, create a render tree(DOM) of elements, compute styles, calculate layout, paint, and render the page. This process ensures proper visual presentation of content.

    - **Parse HTML**: Browser forms a DOM representing content structure.
    - **Load CSS:** Encounters <link> or <style> tags, loads CSS.
    - **Create Render Tree:** Combines DOM and CSS into Render Tree, excluding hidden elements.

- **Apply Styles:** Match elements to CSS rules, calculate styles.
- **Layout (Reflow):** Compute element positions and sizes.
- **Painting:** Apply computed styles, paint elements on screen.
- **Rendering:** Combine painted elements for final user view.

17. Differentiate B/W logical & physical tags

Logical Tags:
- Semantic, structural.
- Define content's meaning and role.
- Examples: <header>, <section>, <footer>.

Physical Tags:
- Presentational.
- Focus on visual appearance.
- Examples: <font>, <b>, <i>.

18. Explain CSS specificity? & how to calculate?

CSS specificity is a rule used by web browsers to determine which style rules should be applied to an element **when multiple conflicting styles are present.** It's a way of resolving conflicts when multiple selectors target the same element with different styles.

CSS Specificity:
- Inline Styles (highest).
- ID Selectors (more specific than classes/elements).
- Class, Attribute, Pseudo-classes.
- Element, Pseudo-elements (least specific).

To calculate specificity:
- Inline styles: 1000 points
- ID selectors: 100 points
- Class, attribute, pseudo-class selectors: 10 points each
- Element, pseudo-element selectors: 1 point each

19. Define gradient in CSS ?

In CSS, a gradient is a **smooth transition between two or more colours,** creating a gradual blend from one color to another. Gradients can be applied as backgrounds, borders, or text colours. They offer design flexibility and depth to elements on a webpage.

Types of gradients in CSS:
- Linear Gradient :
- Radial Gradient

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

background: radial-gradient(
    shape size at
    position, color-stop1, color-stop2, ...);
```

20. Explain CSS Positioning | In Detail

CSS provides several position properties that allow you to control the placement and positioning of elements on a webpage. Here are the different position properties and their possible values:

Position:

static (default): Elements follow the normal document flow.
relative: Positioned relative to its normal position.
absolute: Positioned relative to its closest positioned ancestor or to the containing blocks. It's removed from the normal document flow.
fixed: Positioned relative to the viewport. It remains fixed even when Scrolling.
sticky: It sticks an element to a specific position on the page based on the user's scroll, creating a "sticky" effect

top, right, bottom, left:

Used with position: relative or position: absolute to adjust element's position from its normal or initial position.

z-index:

Specifies the **stacking order** of positioned elements. Elements with higher values appear on top of elements with lower values.

overflow:

Specifies what happens if content **overflows** an element's box. Values include **visible, hidden**, **scroll**, and **auto**.

display:

Controls how an element is displayed. Values include block, inline, inline-block, none, and more.

Float & clear is not recommended

21. How can CSS be integrated into an HTML?

Integrate CSS into HTML:

- **Inline Styles**: Use style attributes on elements.
- **Internal Stylesheet**: Include CSS within <style> in <head>.
- **External Stylesheet**: Link external CSS with <link> in <head>.
- **@import Rule**: In external stylesheet, @import another CSS.

22. What are the advantages and disadvantages of using external style sheets?

Advantages of External Style Sheets:

- **Modularity:** Styles can be reused across multiple pages, ensuring consistency.
- **Centralized Control:** Changes in one place affect all linked pages.
- **Efficiency:** Browser caching reduces page loading time for subsequent visits.
- **Separation of Concerns:** Separates content from presentation, improving maintainability.
- **Faster Updates:** Alter styles without altering HTML content.

Disadvantages of External Style Sheets:

- **External Dependency:** Requires an additional HTTP request, which can marginally impact page load time.
- **Rendering Delay:** External styles must load before content is styled, possibly causing a brief flash of unstyled content (FOUC).
- **Multiple Files:** More files to manage can complicate project structure.
- **Less Suitable for Small Styles:** For tiny projects, inline or internal styles might be more efficient.
- **Limited Offline Access:** May lead to unstyled pages if the external stylesheet isn't cached]

23. Different types of selectors in CSS?

- **Universal Selector (*):** Targets all elements.
- **Type Selector:** Targets elements by tag names (e.g., h1, p).
- **Class Selector (.):** Targets elements by class (e.g., .btn).
- **ID Selector (#):** Targets elements by unique ID (e.g., #header).
- **Attribute Selector ([]):** Targets elements by attributes (e.g., [type="text"]).
- **Pseudo-Class (:):** Targets special states (e.g., :hover, :nth-child(n)).
- **Pseudo-Element (:: or :):** Targets specific parts of an element (e.g., ::before, ::after).
- **Descendant Selector ( ):** Targets nested elements (e.g., ul li).
- **Child Selector (>):** Targets direct children (e.g., ul > li).
- **Adjacent Sibling (+):** Targets element after another (e.g., h2 + p).
- **General Sibling (~):** Targets elements with the same parent (e.g., h2 ~ p).

24. What are the different ways to hide elements using CSS ?

Here are the different ways to hide elements using CSS:

- **display: none;:** Removes element from layout.
- **visibility: hidden;**: Keeps space, but element is invisible.
- **opacity: 0;**: Element becomes transparent.
- **Positioning**: Move **off-screen** using position and top/right/bottom/left.
- **clip-path:** polygon(0 0, 0 0, 0 0, 0 0);: Clips element to hide.
- **z-index: -1;**: Puts element behind others to hide.

25. What does “Cascading” in CSS mean ?

"Cascading" in CSS refers to the **process of combining and applying multiple styles** to an element, where the styles can come from different sources like the browser default styles, user-defined styles, and external stylesheets.

26. Explain the advantages of CSS?

- Separates Style and Content
- Consistent Design
- Easy Updates
- Faster Loading
- Enhances Accessibility
- Responsive Design
- Reusability
- Efficient Maintenance
- SEO Benefits
- Print Styling

27. List out the components of CSS style ?

Here's a concise list of the components of a CSS style:

- **Selectors:** Choose which elements to style.
- **Properties:** Specify visual attributes.
- **Values:** Assign settings to properties.
- **Declarations:** Property-value pairs within curly braces.
- **Rules:** Combine selectors and declarations.

28. Explain "type selectors " in CSS
29. Explain "descendant selector " in CSS
30. Explain “attribute selectors” in CSS
31. Explain " child selectors " in CSS
(Above questions are covered in the question 21)

32. How to use external style sheets?

- **Create CSS File:** Make a separate .css file for your styles.
- **Write Styles:** Define styles in the CSS file.
- **Link CSS to HTML:** In the HTML <head>, add <link> to your CSS file:

  <link rel="stylesheet" type="text/css" href="styles.css">

- **Write HTML:** Add your HTML content in the <body>.
  Separates your styles from your HTML for better organization and reusability.

33. Is the CSS case insensitive?
Yes, by default, CSS is **case-insensitive.**

```css
h1, h2, h3 {
    color: #333;
    font-family: Arial, sans-serif;
}

.button, .link, .nav-link {
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    text-decoration: none;
}
```

34. How to use grouping in CSS ?

How to use grouping in CSS:

- **List Selectors**: Separate selectors with **commas.**
- **Define Styles:** Put shared styles in curly braces.

35. What is “Float” property? & disadvantages
"Float" is a CSS property that aligns elements horizontally. However, it can lead to **layout instability**, **clearing problems**, **collapsed parent containers**, **complexity**, challenges in responsive design, and compromised HTML structure. Modern layout techniques like **Flexbox** and **Grid** are often preferred over "float" due to these disadvantages.

36. List out the media types in CSS

List of CSS media types:

- **all:** All media types (default).
- **braille**: Braille devices.

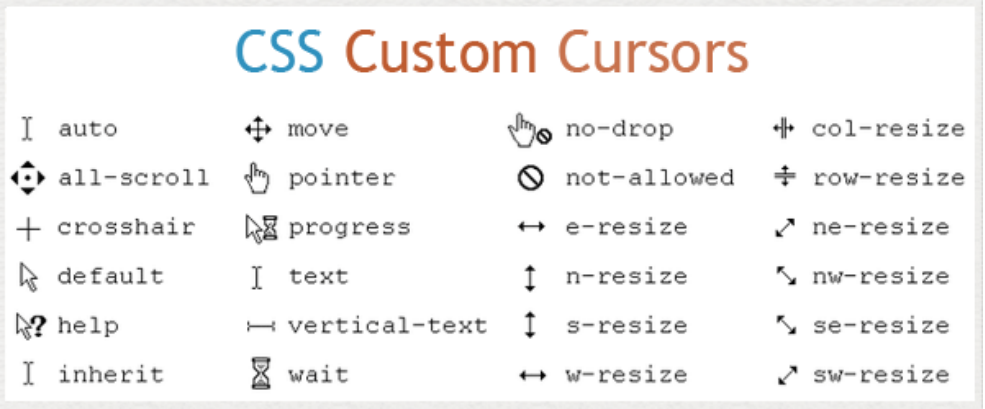
- **embossed:** Paged braille printers.
- **handheld:** Handheld devices.
- **print:** Printed documents.
- **projection**: Projectors.
- **screen:** Screens (computers, tablets, smartphones).
- **speech:** Speech synthesizers.
- **tty:** Fixed-pitch character grid (teletypes, terminals).
- **tv:** Television-type devices.

37. What is the use of z-index in CSS?

The z-index in CSS **controls the layering of elements.** A higher value places an element above others. It's useful for **managing overlapping content** and creating complex layouts.

38. Which property is used to set the cursor pointer?

The **cursor** property in CSS is used to set the appearance of the mouse cursor when it's hovering over an element. **Different Values :**



39. List out properties added in CSS3 ?

<u>More specific use cases/added features in CSS3</u>

- **Layouts: Flexbox** and **Grid** Layout for responsive designs.
- **Effects:** Apply filters (blur, grayscale) using **filter**.
- **Animations:** Create animations with **@keyframes** and animation.
- **Transitions:** Smooth **transitions** with transition.
- **Backgrounds:** Design gradients with **linear-gradient()** & **radial-gradient()**.
- **Typography:** Fine-tune text with text-shadow, font-feature-settings.
- **Responsive:** Adapt styles with **media queries**.
- **Custom Fonts:** Embed fonts with **@font-face.**
- **Rounded Corners:** Rounded elements with **border-radius**.
- **Shadows:** Add depth using **box-shadow**.

40. Difference B/W “display:none;” and “visibility:hidden” in CSS ?

- **display: none;** :
    - Completely removes the element from the layout, including space it occupied.

- **visibility: hidden;** :
    - Hides the element but preserves its space in the layout.

41. What is shadow DOM ?

In simpler terms, shadow DOM lets you create **self-contained components** with their own private "sandboxed" DOM and styles, **reducing conflicts** and making it **easier to create reusable, customizable,** and encapsulated elements on a web page. It's commonly used when building **custom elements or widgets** that need to behave independently and avoid interfering with other parts of the page
Learn in depth…

42. What are pseudo elements & classes?

Pseudo Element :

Pseudo elements are virtual elements in CSS that allow you to **style specific parts of an element's content** without adding extra HTML. They are indicated by double colons (::). Here's a list of common pseudo elements:

- **::before:** Inserts content before an element's content.
- **::after:** Inserts content after an element's content.
- **::first-line:** Styles the first line of text within an element.
- **::first-letter:** Styles the first letter of text within an element.
- **::selection:** Styles the selected text in the browser.
- **::placeholder:** Styles the placeholder text in input elements.
- **::marker:** Styles the marker of a list item.
- **::backdrop:** Styles the backdrop behind a modal or dialog.
- **::cue:** Styles cues in audio and video content.

Pseudo Classes :

Pseudo classes are special keywords in CSS that **target specific states or behaviors** of elements that can't be selected with regular selectors. They are indicated by a single colon (:). Here's a list of common pseudo classes:

- **:hover:** Applies when an element is hovered over.
- **:active:** Applies when an element is clicked or activated.
- **:focus**: Applies when an element gains focus (e.g., through tab navigation).
- **:visited:** Applies to visited links.
- **:nth-child():** Applies to elements based on their position within a parent.
- **:first-child:** Applies to the first child of a parent element.
- **:last-child:** Applies to the last child of a parent element.
- **:not():** Applies when an element does not match a certain selector.
- **:enabled:** Applies to form elements that are enabled.
- **:disabled:** Applies to form elements that are disabled.
- **:checked:** To checked input elements (e.g., checkboxes, radio buttons).
- **:empty:** Applies to elements with no children or text content.
- **:target:** When an anchor link's target matches the current URL fragment.

43. What are Data Attributes ?

Data attributes, in short, are **custom attributes t**hat can be added to HTML elements **to store additional information**. They start with the prefix **"data-"** followed by a descriptive name, allowing JavaScript and CSS to access and manipulate this information without affecting the core functionality.

** John Doe **

In this case, the data-user-id attribute stores the value **"123",** which could be used by JavaScript to perform actions related to that user.

44. Difference B/W display:inline, display:block & display:inline-block;

- **display: inline:**
  Flows within content don't start a new line, **no width/height contro**l.

- **display: block:**
  Starts a new line, takes full width.

- **display: inline-block:**
  Inline flow, **accepts width/height**, respects spacing.

45. What are different breakpoints for different devices ?

- **Mobile Phones:** Typically around 320px to 480px width.
- **Tablets:** Generally from 481px to 1024px width.
- **Laptops/Desktops:** Often 1025px and above.
- **Large Screens:** Above 1440px width (e.g., 4K monitors).

46. Difference B/W “:last-child" and ”list-of-type” Selector

- **:last-child Selector:** Targets the last child element within its parent.
- **:nth-of-type Selector:** Targets elements of a specific type based on their position within their parent.

```css
/* Targeting the last child */
li:last-child {
  color: red;
}

/* Targeting the second item of type li */
li:nth-of-type(2) {
  font-weight: bold;
}
```

47. What is difference B/W em & rem

- **em:** Relative to the font size of the **nearest parent element** with a defined font size.
- **rem:** Relative to the **root (html)** element's font size, providing a consistent base value.

48. How to make Your CSS cross browser compatible ?

- **Follow Standards**: Adhere to W3C standards for consistency.
- **Test on Many Browsers:** Test across various browsers.
- **Use Prefixes**: Add prefixes for experimental features.
- **Reset or Normalize:** Include a reset or normalize stylesheet.
- **Use Media Queries:** Adapt with responsive media queries.
- **Fallbacks:** Offer fallbacks for unsupported features.
- **Be Flexible:** Use relative units for flexibility.
- **Browser-Specific Styles:** Employ conditional styles if needed.
- **Avoid Hacks:** Minimize browser-specific hacks.
- **Stay Updated:** Keep up with evolving standards.

49. Difference b/w mobile first & desktop first

- **Mobile First:** Design approach **starts with mobile screens** and progressively enhances for larger screens. Initially minimal design, then adds complexity.

- **Desktop First:** Design **starts with desktop screens** and adapts downward for smaller screens. Initially complex, then simplifies for smaller screens.

50. Difference b/w bold & strong ?

- **:** Carries strong semantic meaning, often displayed as bold.
- **<b>:** Purely visual, makes text bold without emphasizing importance.

51. What is opacity & why is it used?

Opacity makes elements **transparent**. It's used for **fading effects, overlays**, and **blending with backgrounds** in CSS.

52. What is Viewport Height (vh) & Viewport Width (vw)

- **Viewport Height (vh):** Represents a percentage of the height of the device's viewport. **1vh is 1% of the viewport height.** Useful for responsive vertical sizing.

- **Viewport Width (vw):** Represents a percentage of the width of the device's viewport. **1vw is 1% of the viewport width.** Useful for responsive horizontal sizing.

53. Explain about General sibling selector & adjacent sibling selectors

General Sibling Selector (~): It selects all elements that are siblings of a specified element. It targets elements that **share the same parent and appear after the specified element,** regardless of their hierarchy level.

Adjacent Sibling Selector (+): It selects an element that is **immediately preceded by a specified element.** It targets the first element that directly follows the specified element, sharing the same parent.

Example :

```
<!-- HTML -->

<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

```
/* CSS */

/* General Sibling Selector */
li ~ li {
  color: blue;
}

/* Adjacent Sibling Selector */
li + li {
  font-weight: bold;
}
```

The **li ~ li** selector **targets all li elements that are siblings of another li** (excluding the first one). It turns their text color blue.

The **li + li** selector **targets the second li element,** which directly follows another li element. It makes the second li element bold.

54. What is BEM Method

BEM (Block Element Modifier) is a method for organizing CSS code. It breaks down components into three parts:

- **Block:** The main component or module.**(.button)**
- **Element**: Parts of the block.**(.button_ _text)**
- **Modifier:** Variations or states of the block or element.**(.button--large)**

55. What is a CSS preprocessor?

A CSS preprocessor is a tool that **extends the capabilities of standard CSS** by introducing **variables**, **functions**, **nesting**, and **more**. It allows developers to write cleaner, more organized stylesheets and then compiles them into standard CSS that browsers can understand. Examples :

- Sass/Scss
- Less
- Stylus
- PostCSS

56. What is Bootstrap?

CSS Bootstrap is a **front-end framework** with **ready-made CSS styles**, **components**, and layouts. It speeds up web development by offering a consistent, responsive design system for creating modern websites and apps without starting from scratch.

57. What is a CSS reset?

A CSS reset is a set of CSS rules that aim to **neutralize or reset default browser** styles applied to HTML elements. It's used to establish a consistent baseline for styling across different browsers, ensuring that your **styles start from a clean slate without any unexpected variations** caused by default browser styles.

58. Difference B/W padding & margin

- **Padding** is the **space inside an element,** affecting its content area.
- **Margin** is the **space outside an element,** influencing the spacing between elements in a layout.

59. How to comment in css ? single & multiple line

- **Single-line Comment**: **/*** Your comment ***/** anywhere in CSS.
- **Multi-line Comment**: **/*** Start and end ***/** for spanning lines.

60. Ways to define color in CSS ?

- **Keywords:** Names like red, blue.
- **Hex Codes:** #RRGGBB format, e.g., #FFA500 for orange.
- **RGB**: rgb(R, G, B) format, each value 0-255.
- **RGBA**: rgba(R, G, B, A) format, with alpha (transparency) 0-1.
- **HSL**: hsl(H, S%, L%) format, Hue-Saturation-Lightness.
- **HSLA**: hsla(H, S%, L%, A) with alpha for transparency.

61. Explain about “font-variant” property

The font-variant CSS property modifies how text looks by **altering the font style**. It's used to make specific typographic adjustments within text elements.

For instance:
font-variant: normal; (default) displays text with standard font.
font-variant: **small-caps;** shows uppercase as smaller capitals.

It's useful for **highlighting text,** like titles, using small caps. Keep in mind, **not all fonts support this style,** and results vary based on the chosen font.

62. What are the common CSS properties related to styling tables

- **border-collapse**: Defines whether table borders should be collapsed into a single border or separated.
    - **border-collapse: separate**; (default) - Borders are separate.
    - **border-collapse: collapse**; - Borders are collapsed into a single border.

- **border-spacing:** Specifies the space between adjacent cell borders when border-collapse is set to separate.
    - **border-spacing: 5px 10px;** (horizontal and vertical spacing).

- **table-layout:** Determines the algorithm used to layout table cells, affecting the distribution of column widths.
    - **table-layout: auto;** (default) - Cells size based on content.
    - t**able-layout: fixed;** - Column widths are set by the first row or CSS.

- **caption-side:** Specifies where the table caption (if present) should be positioned.
    - **caption-side: top;** (default) - Caption above the table.
    - **caption-side: bottom;** - Caption below the table.

- **empty-cells**:
  Controls whether or not to display borders and background on empty table cells.
    - **empty-cells:** show; (default) - Show borders and background.
    - **empty-cells: hide;** - Hide borders and background.

- **border:** Sets the border properties for table elements.
    - **border: 1px solid black;**

63. **Use of :not selector ?**
    The **:not** selector in CSS selects elements that **don't match a specific criterion**. It's used to **exclude certain elements from a styling rule**. For instance,

```
/* Select all paragraphs except those with a class of "special" */
p:not(.special) {
    color: blue;
}
```

64. What are the css combinators ?
    CSS combinators are **symbols used to define relationships** between HTML elements when applying styles:

- **Descendant (space)**: Selects elements inside another.
- **Child (>):** Selects direct children of an element.
- **Adjacent sibling (+):** Selects elements immediately after another.
- **General sibling (~):** Selects elements after a specified element with the same parent.

65. Explain “object-fit” & “object-position’ CSS properties

- **object-fit:** Specifies how an image/video should fit within its container while preserving its aspect ratio.
    - **object-fit: fill;** - Stretches to fill container.
    - **object-fit: contain**; - Fits inside container while maintaining aspect ratio.
    - **object-fit: cover;** - Covers container while maintaining aspect ratio.
    - **object-fit: none;** - Keeps original size, possibly overflowing.
    - **object-fit: scale-down**; - Resizes if needed, without exceeding original dimensions.

- **object-position:** Defines the position of an object (image/video) within its container. Example: **object-position: center top;** places object at the top center.

These properties provide control over how media content is displayed and positioned within its container.

66. Explain CSS calc( ) Function

The **calc()** function in CSS allows you to **perform mathematical calculations within property values.** It's particularly useful **for dynamic sizing** and positioning in responsive designs.

For example:

- **width: calc(100% - 20px);** subtracts 20 pixels from 100% width.
- **font-size: calc(14px + 2vmin);** calculates font size based on viewport size.

calc() helps create adaptable and precise styles without resorting to external calculations.

67. What are CSS counters?

CSS counters enable **automatic numbering** or labeling of HTML elements without manual numbering:

- **Creation**: Use **counter-reset** to initialize a counter.
- **Incrementing**: Employ **counter-increment** to increase the counter when specific elements appear.
- **Displaying**: Use content with **::before** or **::after** to show the counter value.
- **Usage**: Apply counters to various HTML elements (**headings, lists**) for automatic numbering or labeling.

```css
ol {
  counter-reset: item;
  list-style-type: none;
}

li::before {
  content: counter(item) ". ";
  counter-increment: item;
}
```

68. What is a CSS grid system?

A CSS grid system is a **layout technique** that uses a grid of rows and columns to **arrange and align elements on a web page.** It simplifies design by providing a structured framework for creating responsive layouts with consistent spacing and positioning.

69. Write down a selector that will match any links ending in .zip, .ZIP, .Zip etc..

To match links that end in .zip, .ZIP, .Zip, etc., you can use the following case-insensitive attribute selector in CSS:

```
a[href$=".zip" i] {
  /* Your styles here */
}
```

The **$=** selector checks if the attribute value ends with a specific string. The **i flag** makes the selector **case-insensitive**, so it will match .zip, .ZIP, .Zip, and other variations.

70. How select every <a> element whose href attribute value begins with "https", .pdf & CSS:

```
a[href^="https"]
a[href$=".pdf"]
a[href*="css"]
```

71. Is there any reason you'd want to use translate() instead of absolute positioning, or vice-versa? And why?

> Use translate() over absolute positioning when:

- **Performance:** translate() is often more efficient for animations because it triggers hardware acceleration, resulting in smoother motion.
- **Simplicity**: It's simpler to use and understand, especially when dealing with responsive layouts.

> Use absolute positioning over translate() when:

- **Precise Placement:** Absolute positioning lets you position elements precisely, regardless of their original position.
- **Z-Index:** It's easier to control the stacking order of elements with absolute positioning and the z-index property.
- **Complex Layouts:** For intricate layouts, absolute positioning provides better control over element placement.

72. Explain ‘Tweening’

In CSS, "**tweening**" involves creating smooth animations by **calculating intermediate states between keyframes**. It ensures gradual transitions for elements, resulting in visually appealing and natural animations.

Example :

Let's say you have a simple animation where you want a square element to smoothly move from the left side of the screen to the right side.

```
@keyframes slide {
  0% {
    transform: translateX(0);
  }
  100% {
    transform: translateX(100%);
  }
}

.square {
  width: 50px;
  height: 50px;
  background-color: blue;
  animation: slide 3s linear infinite;
}
```

In this example, the **@keyframes** rule defines the animation steps. The **.square** class is animated using the slide animation, which moves the square horizontally **from 0% (left)** to **100% (right)** over 3 seconds**. Tweening calculates** the in-between positions, making the movement smooth and continuous.

73. What are the differences between ‘reset’ and ‘normalize’ in CSS?

- **CSS Reset:** Removes all default styles, requiring complete restyling.
- **Normalize CSS:** Preserves useful browser defaults while ensuring consistent styling.

74. What is the property that is used to control image scroll?

The **overflow** property controls image scroll.

75. Explain the concept of CSS Flexbox & its main advantages in web layout design.

CSS Flexbox enables flexible and responsive layouts. It arranges elements in containers along axes, offering:

- **Easy Alignment:** Automatic alignment minimizes complex CSS rules.
- **Responsive:** Adapts smoothly to various screens.
- **Dynamic Sizing:** Elements adjust while maintaining proportions.
- **Order Control:** Easily reorder elements visually.
- **No Hacks:** Eliminates float-based hacks.
- **Efficient Spacing:** Controls space distribution.
- **Nested Layouts:** Ideal for complex structures.
- **Vertical Alignment:** Simplifies vertical centering.

In short, Flexbox optimizes layout, enhancing efficiency and adaptability.

76. How do you vertically center an element within a flex container using Flexbox? Provide the necessary CSS properties and values.

To vertically center an element within a flex container using Flexbox, you can use the **align-items** property with the value **center**. Here's the necessary CSS:

```css
.flex-container {
  display: flex;
  align-items: center; /* Vertically centers items */
}
```

77. Describe the purpose of the flex-grow, flex-shrink, and flex-basis properties in the flex shorthand property. Provide an example

The **flex** shorthand property in CSS Flexbox combines three properties: flex-grow, flex-shrink, & flex-basis.

- **flex-grow:** It defines how much an item can grow compared to other flex items. It accepts a unitless value representing a proportion. An item with a higher value grows more than others.

- **flex-shrink**: It controls the ability of an item to shrink if the container is too small. Similar to flex-grow, it's unitless and indicates the ratio of shrinking for each item.

- **flex-basis:** This sets the initial size of an item before flex-grow and flex-shrink

```
.flex-item {
  flex: 1 0 200px; /* flex-grow: 1, flex-shrink: 0, flex-basis: 200px */
}
```

In this example, **.flex-item** has the following settings:

- **flex-grow:** 1: The item can grow proportionally if there's extra space.
- **flex-shrink:** 0: The item won't shrink to less than its initial size.
- **flex-basis: 200px:** The item starts with an initial size of 200 pixels.

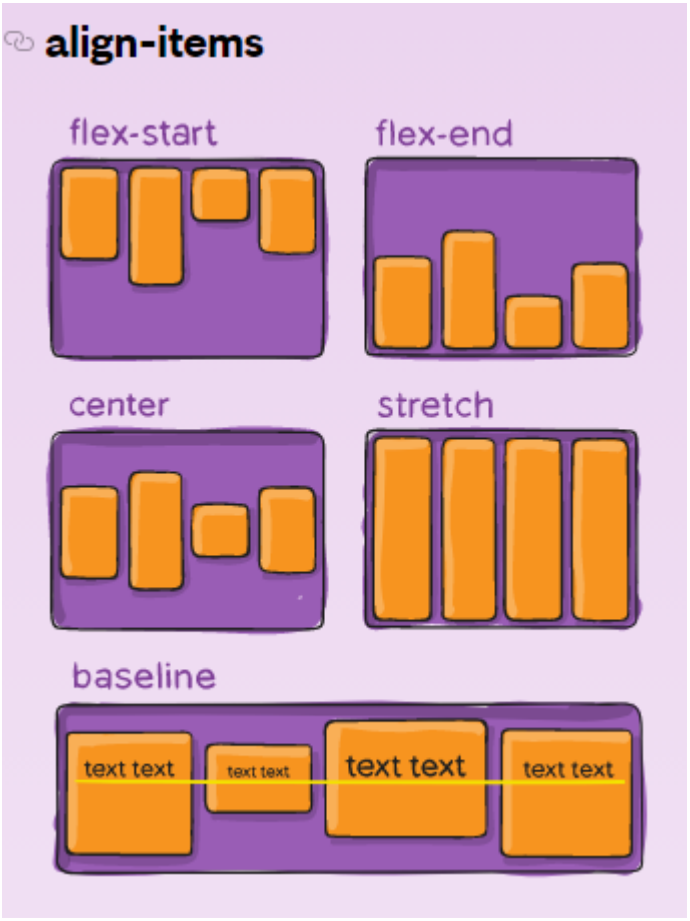
This setup ensures that the item grows to **fill available space** but **doesn't shrink,** maintaining a **minimum width of 200 pixels**. These properties are especially useful for creating flexible layouts where items should distribute space differently while maintaining specific sizing behaviors.

78. In what situations would you use the align-items and align-content properties in a flex container? Provide examples

align-items:

- This property is used to align flex items **individually** along the **cross-axis**.
- It's suitable for scenarios where you want consistent alignment for each **individual item**.
- Example: Vertically centering all items within a flex container.

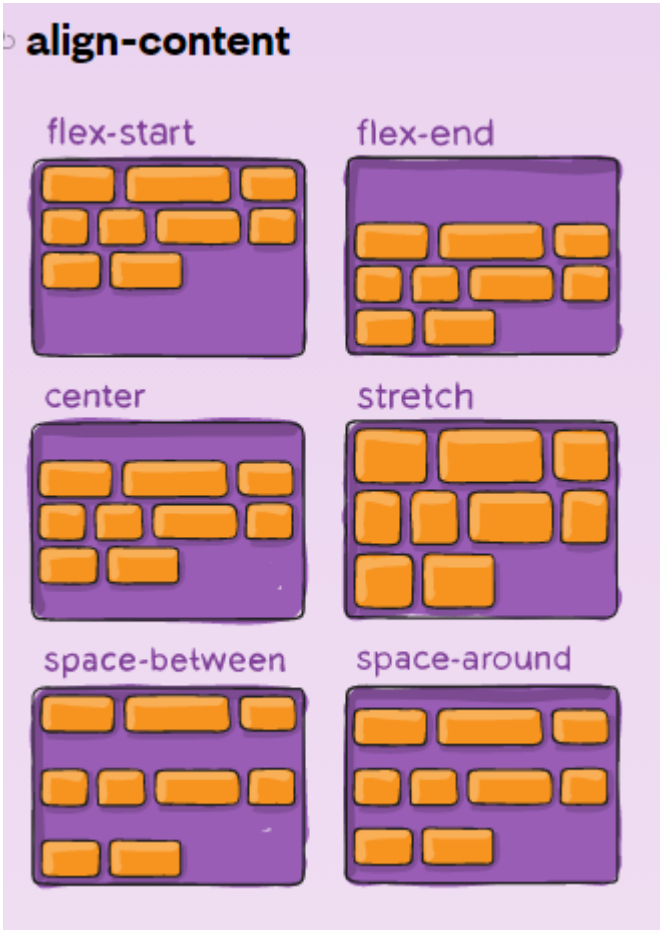
```
.flex-container {
  display: flex;
  align-items: center;
}
```



align-content:

- This property is used to align **lines of flex items** when there's extra space in the cross-axis.
- It's useful when you have **multiple lines of items** in a multi-line flex container.
- Example: Distributing space between lines of items with space between them.

```css
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-between;
}
```



79. Explain the difference between the flex-direction values row and column, and how they influence the arrangement of flex items. Provide a use case for each value.

flex-direction: row: Arranges items horizontally (**left to right** by default).

- Use Case: Navigation menu.

flex-direction: column: Stacks items vertically **(top to bottom**).

- Use Case: Vertical comment feed.

80. Explain the concept of CSS Grid layout and how it differs from other layout systems like Flexbox.

CSS Grid layout is a powerful **two-dimensional** layout system in CSS that allows you to create **complex grid-based layouts with rows and columns**. It offers precise control over both the horizontal and vertical alignment of items.

Flexbox vs. Grid:

- **One-Dimensional vs. Two-Dimensional:** Flexbox handles single-axis arrangements, while Grid manages both rows and columns for intricate layouts.
- **Layout Complexity:** Flexbox suits simple layouts, Grid excels in complex structures with rows and columns.
- **Alignment:** Grid provides advanced alignment control, allowing complex alignments and content justification across the grid.

81. How do you define a grid container and grid items using CSS Grid? Provide an example of a basic grid layout.

1. **Define the Grid Container:**
   Apply **display: grid;** to the parent element to create a grid container.
2. **Define Grid Columns and Rows:**
   Use properties like **grid-template-columns** and **grid-template-rows** to set the size of columns and rows.
3. **Place Grid Items:**
   Use **grid-column** and **grid-row** to specify item positions in the grid.

Example of a basic grid layout:

In this example, the **.grid-container** becomes a grid container with two columns and two rows. **.grid-item** elements are placed in the grid using these columns and rows. The grid-gap adds spacing between items, resulting in a basic grid layout.

82. What are the differences between implicit and explicit grids in CSS Grid? Give an example of when each type might be used.

Implicit Grid:

- Implicit grid tracks are created **automatically** when you place grid items outside the explicitly defined grid using properties like grid-row or grid-column.
- This occurs when items exceed the number of tracks defined in the explicit grid.
- **Use Case:** When items need to overflow and wrap in a grid without having to manually define every track.

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px;
  grid-auto-rows: 50px;
}
```

Explicit Grid:

- Explicit grid tracks are defined using properties like **grid-template-columns** and **grid-template-rows.**
- These tracks are **specified and planned** in advance.
- **Use Case:** When you want precise control over the number and size of tracks in your grid.

```
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: 50px 100px;
}
```

83. Describe the purpose of the grid-template-columns and grid-template-rows properties. Provide examples

The grid-template-columns and grid-template-rows properties in CSS Grid are used to define the sizing and structure of columns and rows within a grid container.

grid-template-columns:

- Specifies the **width of each column** in the grid.
- Accepts values like length **units** (px, fr, etc.), percentages, and auto.
- Example :

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 1fr 2fr;
}
```

grid-template-rows:

- Defines the **height of each row** in the grid.
- Also accepts values like length **units**, percentages, and auto.
- Example :

```
.grid-container {
  display: grid;
  grid-template-rows: 50px auto;
}
```

84. In what scenarios would you use the properties grid-gap, grid-row, and grid-column? Provide examples

- **grid-gap**: Adds spacing between grid items. | `grid-gap: 10px;`
- **grid-row:** Defines vertical span of an item. | `grid-row: 2 / span 2;`
- **grid-column**: Sets horizontal span of an item. | `grid-column: 1 / -1;`

85. How do you create CSS animations? Explain the key components involved in defining animations in CSS

- **@keyframes Rule:** Define animation steps using @keyframes, setting styles at specific points.

- **Animation Property:** Apply animations with the animation property.

  Name: Referenced animation name from @keyframes.
  Duration: Time taken for the animation to complete.
  Timing Function: Animation pace (ease, linear, etc.).
  Delay: Time before animation starts.
  Iteration Count: Number of animation cycles.
  Direction: Forward, backward, alternate, etc.
  Fill Mode: Retaining animation styles before/after.
  Play State: Running or paused.

Example:

```
@keyframes slide {
  0% { transform: translateX(0); }
  100% { transform: translateX(100px); }
}

.element {
  animation: slide 2s ease-in-out infinite alternate;
}
```

86. What is the difference between CSS transitions and CSS animations? Provide examples

CSS Transitions:

- Smoothly **change property values** on user interaction.
- Start and end states, triggered by events like hover.
- Example: Changing width on hover.

```css
.element {
  transition: width 0.3s ease-in-out;
}

.element:hover {
  width: 200px;
}
```

CSS Animations:

- Use **keyframes** for complex, automatic animations.
- Multiple states, easing, delays, and auto-play.
- Example: Sliding element back and forth automatically.

```css
@keyframes slide {
  0% { transform: translateX(0); }
  100% { transform: translateX(100px); }
}

.element {
  animation: slide 2s ease-in-out infinite alternate;
}
```

87. Explain the concept of keyframes in CSS animations. How do you use them to create complex animations?

Keyframes in CSS animations define **specific moments during an animation's** progression. They let you set styles for elements at **different points in time**, making animations controlled and smooth.

<u>Using keyframes for complex animations involves:</u>

- **Defining Keyframes:**
  - Use @keyframes followed by a name and animation progress (0% to 100%).
  - Inside each keyframe, set styles for the element.
- **Intermediate States:**
  - Keyframes create intermediate states between start and end, adding fluidity.
- **Multiple Keyframes:**
  - Combine keyframes at different percentages for intricate animations.

```css
@keyframes bounce {
  0%, 100% { transform: translateY(0); }
  50% { transform: translateY(-50px); }
}

.element {
  animation: bounce 2s ease-in-out infinite;
}
```

88. What is the significance of the animation property in CSS? Describe its components and how they influence the behavior of an animation.

The **animation** property in CSS is a **shorthand** property that **combines** several individual **animation-related properties** to define the behavior of an animation applied to an Element.

Property	Description	Influence
Animation Name	Specifies the name of defined @keyframes animation.	Determines animation to apply.
Animation Duration	Sets time for animation to complete one cycle.	Influences animation speed.
Animation Timing Function	Defines pace of animation's progress (ease, linear, etc.).	Controls acceleration and deceleration.
Animation Delay	Determines time before animation starts.	Influences animation start time.
Animation Iteration Count	Specifies how many times animation should repeat.	Sets number of animation cycles.
Animation Direction	Sets animation direction (normal, reverse, alternate).	Controls playback direction.
Animation Fill Mode	Defines element's appearance before/after animation.	Manages pre/post-animation styles.
Animation Play State	Determines if animation is running or paused.	Influences animation's active state.

89. How can you optimize CSS animations for performance? What techniques or best practices would you consider to ensure smooth animations without causing performance bottlenecks?

Optimize CSS animations for performance:

- **Use Hardware Acceleration:** Apply animations to properties like transform and opacity for smoother rendering using the GPU.
- **Minimize DOM Manipulation:** Limit changes to animated elements to reduce layout recalculations.
- **Reduce Animation Complexity:** Avoid heavy animations involving many elements or properties.
- **Use Debouncing and Throttling:** Control animation triggers to prevent excessive updates.
- **Avoid Long Durations:** Shorter durations provide more responsive animations.
- **Use will-change:** Declare animation-intensive elements for browser optimization.
- **Optimize Images:** Use appropriate image formats and sizes to reduce load times.
- **Use CSS Transitions:** Use transitions for simpler animations, as they often perform better.

90. What are CSS media queries, and how do they work? Provide an example

CSS media queries are a feature that allows you to apply styles based on the characteristics of the **user's device or screen**. They enable you to create responsive designs that adapt to **different screen sizes, orientations,** and other attributes.

Media queries work by evaluating the conditions you specify and applying the **associated styles** if those conditions are met.

```css
/* Default styles for all screen sizes */
.element {
  font-size: 16px;
}

/* Media query for screens narrower than 600px */
@media (max-width: 600px) {
  .element {
    font-size: 14px;
  }
}

/* Media query for screens wider than 1200px */
@media (min-width: 1200px) {
  .element {
    font-size: 18px;
  }
}
```

91. Explain the concept of breakpoints in the context of media queries. How do you decide on appropriate breakpoints for different devices?

Breakpoints in the context of media queries **refer to specific screen widths** at which you **adjust your styles** to create a responsive design. They are the points where your layout needs to adapt to fit different devices or screen sizes.

Deciding on breakpoints:

- **Device Research:** Know common sizes and trends for devices.
- **Content and Design:** Adapt for readability and usability.
- **User Behavior:** Analyze popular screen sizes.
- **Testing:** Identify content issues through testing.
- **Responsive Frameworks:** Adjust based on framework's breakpoints.

92. What is the difference between min-width and max-width in media queries?

min-width:

- Specifies the minimum width at which the styles should apply.
- Styles will be applied when the viewport width is equal to or greater than the specified value.
- Example: @media (min-width: 768px) { /* Styles here */ }

max-width:

- Specifies the maximum width at which the styles should apply.
- Styles will be applied when the viewport width is equal to or smaller than the specified value.
- Example: @media (max-width: 1024px) { /* Styles here */ }

In essence, **min-width** targets larger screens, while **max-width** targets smaller screens.

93. How can you use media queries to target different devices, such as smartphones, tablets, and desktops?

- Smartphones (up to 767px)
- Tablets (768px to 1023px)
- Desktops (1024px and above)

94. What is the purpose of the meta viewport tag in responsive web design? How does it relate to media queries, and why is it important for mobile devices?

The **meta viewport** tag in responsive web design is a crucial element that controls how a web page is displayed on different devices, particularly mobile devices. It provides instructions to the browser regarding the initial zoom level and how content should be scaled to fit the viewport.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- **width=device-width** ensures the website's width matches the device's width,
- **initial-scale=1.0** prevents the page from being zoomed in initially. This helps ensure the website is responsive and readable on mobile devices.

95. What are CSS transitions and how do they work? Provide an example

CSS transitions allow smooth property changes over time. They work by defining a **property**, **duration**, **timing**, and **event**. When triggered (e.g., on hover), the property gradually changes to the new value.

```css
/* Initial state */
.element {
  width: 100px;
  background-color: blue;
  transition: width 0.5s ease-in-out;
}

/* Hover state */
.element:hover {
  width: 200px;
  background-color: red;
}
```

In this example, when you hover over the .element:

- The **width** property transitions smoothly from 100px to 200px over 0.5 seconds.
- The **background-color** property also changes from blue to red.
- The **ease-in-out** timing function ensures a smooth acceleration and deceleration effect during the transition.

96. How can you transition multiple properties simultaneously using CSS? Provide an example of transitioning both opacity and transform properties.

You can transition multiple properties simultaneously by listing them within the transition property, separated by commas.

Here's an example of transitioning both the opacity and transform properties:

```css
.element {
  opacity: 0.5;
  transform: scale(1);
  transition: opacity 0.3s ease-in-out, transform 0.3s ease-in-out;
}

.element:hover {
  opacity: 1;
  transform: scale(1.2);
}
```

In this example:

- The **opacity** property changes smoothly from 0.5 to 1.
- The **transform** property transitions smoothly from its initial state to a scaled state (1 to 1.2) on **:hover**.
- Both transitions have a duration of 0.3 seconds and use the ease-in-out timing function.

97. Explain the concept of event-triggered transitions. How do you use pseudo-classes like :hover to create interactive transitions?

Event-triggered transitions are CSS transitions that are activated by specific events, such as **hovering over** an element or **clicking it**. These transitions enable you to create interactive effects that enhance user experience without the need for JavaScript.

```css
/* Initial state */
.element {
  opacity: 0.7;
  transition: opacity 0.3s ease-in-out;
}

/* Hover state */
.element:hover {
  opacity: 1;
}
```

In this example, when you hover over the **.element:**

- The property transitions smoothly from 0.7 to 1 over 0.3 seconds.
- This creates a fade-in effect when the user interacts with the element.

By combining event-triggered pseudo-classes like **:hover** with transitions, you can add interactivity and engagement to your designs.

98. What are CSS variables (custom properties)? How do they differ from traditional variables in programming languages?

CSS variables, also known as custom properties, are a way to store and reuse values in CSS. They allow you to define a value once and reuse it throughout your stylesheet

CSS variables are defined using the **--** prefix followed by a name, like **--main-color** or **--font-size**.

CSS Variables vs. Traditional Programming Variables:

Aspect	CSS Variables	Traditional Programming Variables
Scope	Scoped to elements or globally using :root	Often block or function scope
Dynamic Nature	Dynamic, can change with JavaScript	Assigned constant values
Property Usage	Style property values	Various data types
Calculation and Transformation	Used in property calculations	Operate based on data type
Interpolation	Direct interpolation using var()	Concatenation or formatting may be needed
Fallbacks	Define fallbacks for unsupported browsers	Fallbacks may vary by language

99. **What font-related CSS properties are used to control the appearance of text?**

- **font-size:** Sets the size of the font.
- **font-family:** Specifies the font(s) to be used for text content.
- **font-weight:** Defines the thickness or boldness of the font.
- **font-style:** Specifies the style of the font (normal, italic, or oblique).
- **font-variant:** Controls the use of small-caps characters in the font.
- **line-height:** Sets the space between lines of text.
- **letter-spacing:** Adjusts the space between characters.
- **word-spacing:** Controls the spacing between words.
- **text-align:** Determines the horizontal alignment of text within its container.

- **text-decoration:** Adds visual styling (underline, overline, line-through) to text.
- **text-transform:** Modifies the capitalization of text (uppercase, lowercase, capitalize).
- **text-shadow:** Applies a shadow effect to the text.
- **color:** Sets the color of the text.

100. What is the @font-face rule in CSS? How do you use it to include custom fonts in your web pages?

The **@font-face** rule in CSS is used to **define custom fonts** that can be used on a web page. It allows you to include **external font files** and use them in your styles.

This is particularly useful when you want to use non-standard fonts that may not be available on all users' devices.

The @font-face rule consists of several descriptors:

- **font-family:** Specifies the name you'll use to refer to the font in your styles.
- **src:** Defines the source of the font files. This includes URLs to font files like WOFF, WOFF2, TTF, or EOT formats.
- **font-weight:** Specifies the font weight (e.g., normal, bold).
- **font-style:** Defines the font style (e.g., normal, italic).
- **font-display:** Specifies how the font should be displayed while it's loading.
- **unicode-range:** Allows you to specify a range of characters that should be included in the font.

Example usage of @font-face:

```
@font-face {
  font-family: 'CustomFont';
  src: url('customfont.woff2') format('woff2'),
       url('customfont.woff') format('woff');
  font-weight: normal;
  font-style: normal;
}
```

With **@font-face,** you can use web fonts that aren't installed on users' systems, ensuring consistent typography across different devices and platforms.