**CSS Units & Responsive Design , Transitions, Hover effects**

**1. CSS Units**

CSS uses different **units to define sizes, spacing, fonts, widths, heights, etc.**. Choosing the right unit is key for **responsive layouts**.

**A. Absolute Units**

| **Unit** | **Description** | **Usage** |
| --- | --- | --- |
| px | Pixels; fixed unit | Borders, small text, icons |
| pt | Points (1pt ≈ 1.333px) | Printing, rarely web |
| cm, mm, in | Physical units | Rarely used on screens |

**Example:**

div {

width: 300px;

height: 150px;

border: 2px solid black;

}

* **Pixels** = fixed → does not scale on different screen sizes.

**B. Relative Units**

| **Unit** | **Description** | **Usage** |
| --- | --- | --- |
| % | Percentage relative to **parent element** | Width, height, padding, margin |
| em | Relative to **font-size of parent** | Fonts, spacing |
| rem | Relative to **font-size of root (html)** | Fonts, spacing, layout |
| vh | 1% of viewport **height** | Full-screen sections |
| vw | 1% of viewport **width** | Full-width elements, typography |

**Examples:**

/\* Percentage relative to parent \*/

div {

width: 50%; /\* half of parent width \*/

padding: 10%; /\* 10% of parent width \*/

}

/\* em relative to parent font-size \*/

p {

font-size: 1.5em; /\* 1.5 times parent font size \*/

}

/\* rem relative to root font-size \*/

h1 {

font-size: 2rem; /\* 2 times root font size (default 16px -> 32px) \*/

}

/\* Viewport units \*/

section {

height: 100vh; /\* full viewport height \*/

width: 100vw; /\* full viewport width \*/

}

**Tips:**

* Use **px** for fixed precision, **em/rem** for scalable fonts.
* Use **%** for container-based sizing.
* Use **vh/vw** for **full-screen layouts and responsive typography**.

**2. Responsive Web Design Basics**

**Goal:** Make a website look **good on all screen sizes** (desktop, tablet, mobile).

**Techniques:**

1. **Fluid layouts** → % / em / rem units
2. **Flexible images** → max-width: 100%
3. **Media Queries** → apply styles for specific screen widths

**A. Media Queries**

* Apply CSS rules **only for specific screen sizes**.
* Syntax:

@media (condition) {

/\* CSS rules \*/

}

**Examples:**

1. **Mobile-first approach (max-width):**

/\* Default styles for desktop \*/

body {

font-size: 16px;

}

/\* Styles for devices ≤ 600px \*/

@media (max-width: 600px) {

body {

font-size: 14px;

}

}

1. **Tablet-specific layout**

@media (min-width: 601px) and (max-width: 1024px) {

.container {

width: 80%;

}

}

1. **Landscape vs Portrait**

@media (orientation: portrait) {

body {

background-color: lightblue;

}

}

**B. Responsive Images**

img {

max-width: 100%;

height: auto;

}

* Keeps image inside its container without distortion.

**C. Example: Responsive Layout**

<div class="container">

<p>Responsive Box</p>

</div>

.container {

width: 70%; /\* fluid width \*/

padding: 20px;

background: lightgreen;

}

@media (max-width: 768px) {

.container {

width: 90%; /\* expand on smaller screens \*/

padding: 10px;

}

}

@media (max-width: 480px) {

.container {

width: 100%; /\* full width on mobile \*/

padding: 5px;

}

}

**Explanation:**

* Container adjusts width based on screen size.
* Padding also adjusts for better spacing on smaller devices.

**3. Quick Tips / Interview Points**

1. **px** → fixed size, not scalable
2. **em** → relative to **parent font-size**, good for nested elements
3. **rem** → relative to **root font-size**, easier for global scaling
4. **%** → relative to parent → fluid layout
5. **vh/vw** → relative to viewport → full-screen sections
6. **Media Queries** → essential for responsive design
7. **Mobile-first approach** → write default styles for mobile, then larger screens
8. Use **flexbox/grid + relative units + media queries** → fully responsive layout

**4. Common Interview Questions**

1. Difference between **px, em, rem, %, vh, vw**?
2. When to use **em vs rem**?
3. Difference between **max-width and min-width in media queries**?
4. How to make an image **responsive**?
5. What is **mobile-first vs desktop-first** approach?
6. How do **viewport units** differ from percentage?

**Transitions, Hover effects**

**What are CSS Transitions?**

A **CSS Transition** is used to create smooth **animation-like effects** when a property changes from one value to another.

Instead of changing instantly, transitions make the change happen **gradually** over a specific duration.

When you hover over a button, instead of immediately changing color, it fades smoothly.

**Basic Syntax**

selector {

transition: property duration timing-function delay;

}

**🔹 Example**

button {

background-color: purple;

color: white;

padding: 10px 20px;

transition: background-color 0.5s ease, transform 0.3s;

}

button:hover {

background-color: black;

transform: scale(1.1);

}

**Explanation:**

* When hovered → button color changes from purple to black **smoothly** over 0.5 seconds.
* It also **zooms slightly** using transform: scale(1.1).

**Transition Properties**

| **Property** | **Description** |
| --- | --- |
| transition-property | Specifies which CSS property to animate (like color, width, background, transform). |
| transition-duration | Defines how long the transition takes (e.g., 0.5s, 2s). |
| transition-timing-function | Defines how the speed of the transition changes (ease, linear, ease-in, ease-out). |
| transition-delay | Delays the start of the transition. |

**2. Hover Effects**

A **hover effect** occurs when the mouse pointer moves over an element — it changes style dynamically.

Usually combined with transitions to make animations smooth.

**Example – Button Glow**

button {

background: purple;

color: white;

border: none;

padding: 10px 20px;

border-radius: 8px;

transition: all 0.3s ease;

}

button:hover {

background: violet;

box-shadow: 0 0 15px violet;

transform: scale(1.05);

}

The button glows and enlarges softly when hovered.

**Example – Image Zoom Effect**

<div class="img-box">

<img src="image.jpg" alt="sample">

</div>

.img-box img {

width: 250px;

border-radius: 10px;

transition: transform 0.6s ease;

}

.img-box img:hover {

transform: scale(1.2);

}

When hovered, the image zooms in slightly — looks dynamic!

**Combining Transitions + Hover = Modern UI Magic**

Transitions + Hover together create most of the **interactive UI** effects we see daily:

* Buttons glow or bounce
* Cards move upward
* Nav links underline smoothly
* Icons rotate or fade in

**1. What is a Pseudo-Class?**

A **pseudo-class** is used to style an element based on its **state, action, or position** — even if that state isn’t directly written in HTML.

**Hover effect**

<button>Click Me</button>

button {

background: purple;

color: white;

padding: 10px 20px;

border: none;

}

button:hover {

background: black;

transform: scale(1.1);

}

When you move your mouse over the button . it turns black and slightly enlarges.

**Focus effect**

<input type="text" placeholder="Enter your name">

input:focus {

border: 2px solid purple;

outline: none;

box-shadow: 0 0 5px purple;

}

When you click inside the input box .border glows purple.

**Table highlighting**

tr:nth-child(even) {

background: #f3e5f5;

}

Every even row of your table gets a light purple color automatically.

**Use in Real Websites**

* Highlight navigation menus on hover
* Change form field color when active
* Style first or last items in lists
* Animate buttons when clicked

**2. What is a Pseudo-Element?**

A **pseudo-element** lets you style **specific parts** of an element or **insert virtual content** without actually adding it to HTML.  
Think of it as adding “imaginary elements” inside your tag using CSS.

**Add icons using ::before**

<h2>Dashboard</h2>

h2::before {

content: "📋 ";

}

➡ Adds an icon before the text "Dashboard" without changing HTML.

**Add decoration line using ::after**

h2::after {

content: "";

display: block;

width: 100px;

height: 3px;

background: purple;

margin-top: 5px;

}

Adds a stylish underline below the heading.

**Style first letter**

p::first-letter {

font-size: 35px;

color: purple;

font-weight: bold;

}

Makes the first letter of a paragraph stand out .

**Use in Real Websites**

* Decorative effects (lines, icons)
* Adding quotes automatically
* Highlighting first letter or line of articles
* Styling user-selected text

**3. Difference Between Pseudo-Class and Pseudo-Element**

| **Feature** | **Pseudo-Class** | **Pseudo-Element** |
| --- | --- | --- |
| Controls | State or condition | Specific part of element |
| Symbol | : single colon | :: double colon |
| Example | a:hover, input:focus | p::first-letter, div::after |
| Real or virtual | Acts on real element | Creates virtual content |
| Used For | Hover, Focus, Active, nth-child | Before, After, First-letter, Selection |

**4. What is Position Property in CSS?**

The **position property** defines **how an element is placed** on a webpage — whether it stays in flow, moves freely, or sticks to the screen.

**🔹 Syntax**

position: value;

**🔸 1. Static (Default)**

All elements are **static by default** — they appear in the normal document flow.

div {

position: static;

}

You can’t move it using top/left/bottom/right.  
Example: Normal paragraph or div without special positioning.

**🔸 2. Relative**

Moves the element **relative to its original place**.

div {

position: relative;

top: 20px;

left: 30px;

}

It shifts slightly but still keeps its original space reserved.

**Use:** Small adjustments (like moving text or icon a few pixels).

**🔸 3. Absolute**

Moves element **out of normal flow** and positions it relative to the **nearest ancestor** that has position set (not static).

.parent {

position: relative;

}

.child {

position: absolute;

top: 20px;

right: 30px;

}

**Use:** Tooltips, dropdown menus, popups, badges on icons.

**🔸 4. Fixed**

Stays **fixed to the viewport** — does not move even when you scroll.

.chatbox {

position: fixed;

bottom: 20px;

right: 20px;

}

**Use:** “Back to top” buttons, sticky chat icons, floating menus.

**🔸 5. Sticky**

Acts **relative until scrolling**, then becomes **fixed** when reaching a certain position.

header {

position: sticky;

top: 0;

background: white;

}

**Use:** Sticky navbar that stays at top while scrolling.

**5. Z-Index Property**

z-index decides which element appears **on top or behind** another (like layers in Photoshop).

**🔹 Syntax**

z-index: number;

Higher value = element appears in front.

Works only when position is **relative, absolute, fixed, or sticky**.

**Overlapping Boxes**

**Index.html**

<div class="box box1">Box 1</div>

<div class="box box2">Box 2</div>

<div class="box box3">Box 3</div>

**Style.css**

.box {

width: 150px;

height: 150px;

position: absolute;

text-align: center;

line-height: 150px;

font-weight: bold;

color: white;

}

.box1 { background: purple; top: 40px; left: 40px; z-index: 1; }

.box2 { background: violet; top: 70px; left: 70px; z-index: 2; }

.box3 { background: black; top: 100px; left: 100px; z-index: 3; }

“Box 3” appears at the top, because it has the highest z-index.

**Real Website Use Cases**

* Popup windows over background
* Navigation bars above images
* Sticky chat icons above content
* Modal dialogs or alert boxes