DAY-3 : CSS Advanced Topics

---

1. CSS Variables (Custom Properties)

CSS variables let you define reusable values that can be changed dynamically.
They start with -- and are accessed using var().
Example: --main-color: blue; and use it with color: var(--main-color);.
They reduce code duplication by centralizing style values.
Variables can be declared globally under :root or scoped to an element.
They support inheritance, meaning children can use parent variables.
Useful for theming (light mode, dark mode).
You can dynamically update variables with JavaScript.
They make maintaining large projects much easier.
They are now widely supported in all major browsers.

1) CSS Variables (Custom Properties)

```
:root{
  --main-color: #2dd4bf;
  --surface: #111827;
  --radius: 12px;
  --gap: 1rem;
}

/* Use variables */
.button{
  color: var(--main-color);
  background: var(--surface);
  border-radius: var(--radius);
  padding: var(--gap);
}

/* Scoped override */
.theme-dark{ --surface: #0b1220; }
```

2. CSS Functions (calc(), min(), max(), clamp())

CSS provides built-in math functions to create flexible layouts.
calc() lets you perform calculations inside CSS (e.g., width: calc(100% - 50px)).
min() picks the smallest value, useful for preventing oversized elements.
max() ensures elements don't shrink below a certain size.
clamp() combines min, preferred, and max values (clamp(200px, 50%, 800px)).
They improve responsiveness without complex media queries.
They work with all CSS units like %, px, em, rem.
Best used in typography, layouts, and spacing.
They reduce the need for JavaScript in simple calculations.
They help balance between flexibility and control.

2) CSS Functions: calc(), min(), max(), clamp()

```
.container{ width: calc(100% - 2rem); }


.card{ max-width: min(72ch, 92vw); }


.sidebar{ width: max(240px, 20vw); }


h1{ font-size: clamp(1.25rem, 4vw, 2.25rem); }


.grid{
  grid-template-columns: repeat(auto-fit, minmax(clamp(12rem, 30vw, 20rem), 1fr));
}
```

3. Advanced Positioning (sticky, fixed, relative vs absolute)

CSS offers multiple positioning strategies for elements.
static is the default, meaning elements follow normal flow.
relative shifts an element without removing it from the flow.
absolute positions an element relative to the nearest positioned ancestor.
fixed locks an element to the viewport (e.g., sticky navbars).
sticky combines relative + fixed; it sticks within its parent container.
Positioning helps build overlays, modals, and custom layouts.

You must consider stacking and z-index when layering positioned elements.
Improper use may cause layout breakages.
Mastery of positioning is essential for professional front-end work.

3) Advanced Positioning (relative, absolute, fixed, sticky)

.rel{ position: relative; top: 0.25rem; left: 0.25rem; }

.abs-parent{ position: relative; }

.abs-child{ position: absolute; top: .5rem; right: .5rem; }

.fixed-top{ position: fixed; top: 0; left: 0; right: 0; height: 3rem; }

.sticky-box{ position: sticky; top: 4rem; }

.static-default{ position: static; } /* default */

4. CSS Z-index & Stacking Context

z-index controls the order of overlapping elements.
Higher values appear above lower ones.
A stacking context is created when certain CSS properties are applied (e.g., position: relative; opacity < 1; transform).
Not all elements respond to z-index unless they belong to the same stacking context.
Common use cases: dropdown menus, modals, tooltips.
Misusing z-index can cause invisible click-blocking elements.
Debugging often requires developer tools to inspect stacking contexts.
Best practice: use minimal values (like 1–10) and avoid very high numbers.
Organize z-index layers logically for scalability.
Understanding stacking is crucial for UI/UX smoothness.

4) Z-index & Stacking Context

.dropdown{ position: relative; z-index: 10; }

.backdrop{ position: fixed; inset: 0; z-index: 900; }

.modal{ position: fixed; inset: auto 0 0 0; z-index: 1000; }

```
/* New stacking context examples */

.layer-transform{ transform: translateZ(0); } /* creates context */

.layer-opacity{ opacity: .999; }            /* creates context */
```

5. CSS Filters & Blend Modes

CSS filters apply graphical effects like blur, grayscale, or brightness.
Example: filter: blur(5px) brightness(80%);.
Blend modes (mix-blend-mode) define how elements mix with backgrounds.
They mimic Photoshop-like effects directly in CSS.
Common blend modes: multiply, screen, overlay, difference.
Useful for image effects, backgrounds, and creative designs.
They can enhance UI aesthetics without editing images.
Performance impact is minimal compared to animations.
Supported across major browsers.
Best combined with gradients or images for rich visual design.

5) Filters & Blend Modes

```
/* Filters */

.thumb{ filter: grayscale(100%) contrast(110%); }

.thumb:hover{ filter: none; }


/* Blend with background */

.overlay{ background: #0ea5e9; mix-blend-mode: multiply; }


/* Backdrop blur (needs support) */

.glass{

  background: rgba(255,255,255,.1);

  -webkit-backdrop-filter: blur(8px);

        backdrop-filter: blur(8px);

}
```

6. CSS Shapes & Clip-path

Clip-path defines custom shapes for elements, beyond rectangles.
You can create circles, polygons, and complex masks.
Example: clip-path: circle(50% at center);.
It controls which part of an element is visible.
Great for profile images, creative buttons, and web art.
CSS Shapes allow text to flow around non-rectangular images.
This enhances magazine-style or editorial layouts.
Shapes can be animated for dynamic effects.
Browser support is growing but not universal.
They bring design creativity without heavy graphics.

6) CSS Shapes & clip-path

```
/* Clip element */
.avatar{
  width: 160px; height: 160px;
  clip-path: circle(50% at 50% 50%);
}

.ribbon{
  clip-path: polygon(0 0, 100% 0, 85% 100%, 0 100%);
}

/* Flow text around a shape */
.figure{
  float: left; width: 240px; height: 240px; border-radius: 50%;
  shape-outside: circle(50%);
}
```

7. CSS Masking & Gradients

CSS masking hides parts of an element using an image or gradient mask.
Gradients (linear, radial, conic) allow smooth color transitions.
Example: background: linear-gradient(to right, red, blue);.
Radial gradients spread from a central point.

Conic gradients rotate colors around a center.
Masks allow advanced transparency effects.
Used in banners, backgrounds, and UI cards.
They reduce dependency on external graphics.
Blend masks with animations for futuristic effects.
Combined, gradients and masks power modern design trends.

7) Masking & Gradients

```
/* Linear / radial / conic gradients */
.banner{
  background:
    linear-gradient(135deg, #0ea5e9, #22d3ee);
}
.spot{
  background: radial-gradient(circle at 30% 40%, #22d3ee33, transparent 60%);
}
.wheel{
  background: conic-gradient(#10b981 0 120deg, #f59e0b 120deg 240deg, #ef4444 240deg 360deg);
}

/* Masks */
.fade-bottom{
  -webkit-mask: linear-gradient(#000 70%, transparent 100%);
          mask: linear-gradient(#000 70%, transparent 100%);
}
.logo-cutout{
  -webkit-mask-size: cover; mask-size: cover;
  -webkit-mask-repeat: no-repeat; mask-repeat: no-repeat;
  /* -webkit-mask-image / mask-image set to an image or gradient elsewhere */
}
```

8. CSS Units (em, rem, vh, vw, %)

CSS supports absolute and relative units.
px is fixed, while % depends on parent size.
em is relative to parent font-size, while rem is relative to root.
vh and vw represent viewport height and width percentages.
They ensure designs adapt across screen sizes.
Best practice: use rem for consistent typography scaling.
Combine % with calc() for flexible layouts.
Viewport units are perfect for fullscreen sections.
Mixing units requires testing across devices.
Choosing the right unit avoids accessibility issues.

8) Units (em, rem, vh, vw, %)

```
html{ font-size: 16px; } /* 1rem = 16px */


h1{ font-size: 2rem; }      /* root-relative */

.card{ padding: 1.25rem; }


.item{ font-size: 1em; }    /* parent-relative */

.item .icon{ width: 1.5em; }


.hero{ height: 60vh; }      /* viewport height */

.section{ width: 90vw; }    /* viewport width */


.fluid{ width: 80%; }       /* parent-relative */
```

---

9. CSS Responsive Images (object-fit, picture, srcset)

Modern CSS + HTML support responsive images.
object-fit ensures images fit containers without distortion.
cover, contain, and fill are commonly used values.
picture and srcset allow different images for different devices.

This optimizes performance by loading smaller files on mobile.
Great for responsive hero images and product displays.
Avoids stretching or squashing images.
Critical for high-DPI screens like Retina displays.
Enhances SEO and Core Web Vitals by improving load time.
Combines with media queries for perfect results.

9) Responsive Images (CSS side for object-fit)

```
.cover{
  width: 100%;
  height: 240px;
  object-fit: cover;        /* keep aspect, fill box */
  object-position: center;
}

.contain{
  width: 100%;
  height: 240px;
  object-fit: contain;      /* keep aspect, fit inside */
  background: #111827;
}
```

(Note: picture/srcset/sizes are HTML attributes; CSS shown here only.)

10. CSS Best Practices & Performance

Keep CSS modular and organized with classes, not IDs.
Avoid deeply nested selectors; they slow rendering.
Minify and combine CSS files for performance.
Use shorthand properties (e.g., margin, padding).
Leverage browser dev tools to debug layouts.
Avoid inline CSS for maintainability.
Test on multiple devices for cross-browser compatibility.
Lazy-load only required CSS for faster initial loads.

Follow naming conventions like BEM for scalability.
Prioritize accessibility with readable font sizes and contrasts.

Perfect—here are **CSS-only** “syntax blocks” for each advanced topic (no JS, no HTML). Copy-paste as needed.

10) Best Practices & Performance

```
/* BEM */
.card{}
.card__title{}
.card__action{}
.card--featured{}

/* Low specificity */
.nav__item > .link{}

/* Motion preferences */
@media (prefers-reduced-motion: reduce){
  *{ animation-duration:.001ms !important; transition-duration:.001ms !important; }
}

/* Logical properties (i18n-friendly) */
.panel{
  padding-block: 1rem;   /* top/bottom */
  padding-inline: 1rem;  /* left/right */
}

/* Utility tokens */
:root{ --z-dropdown: 10; --z-overlay: 1000; }
.dropdown{ z-index: var(--z-dropdown); }
```

```
.modal{ z-index: var(--z-overlay); }
```