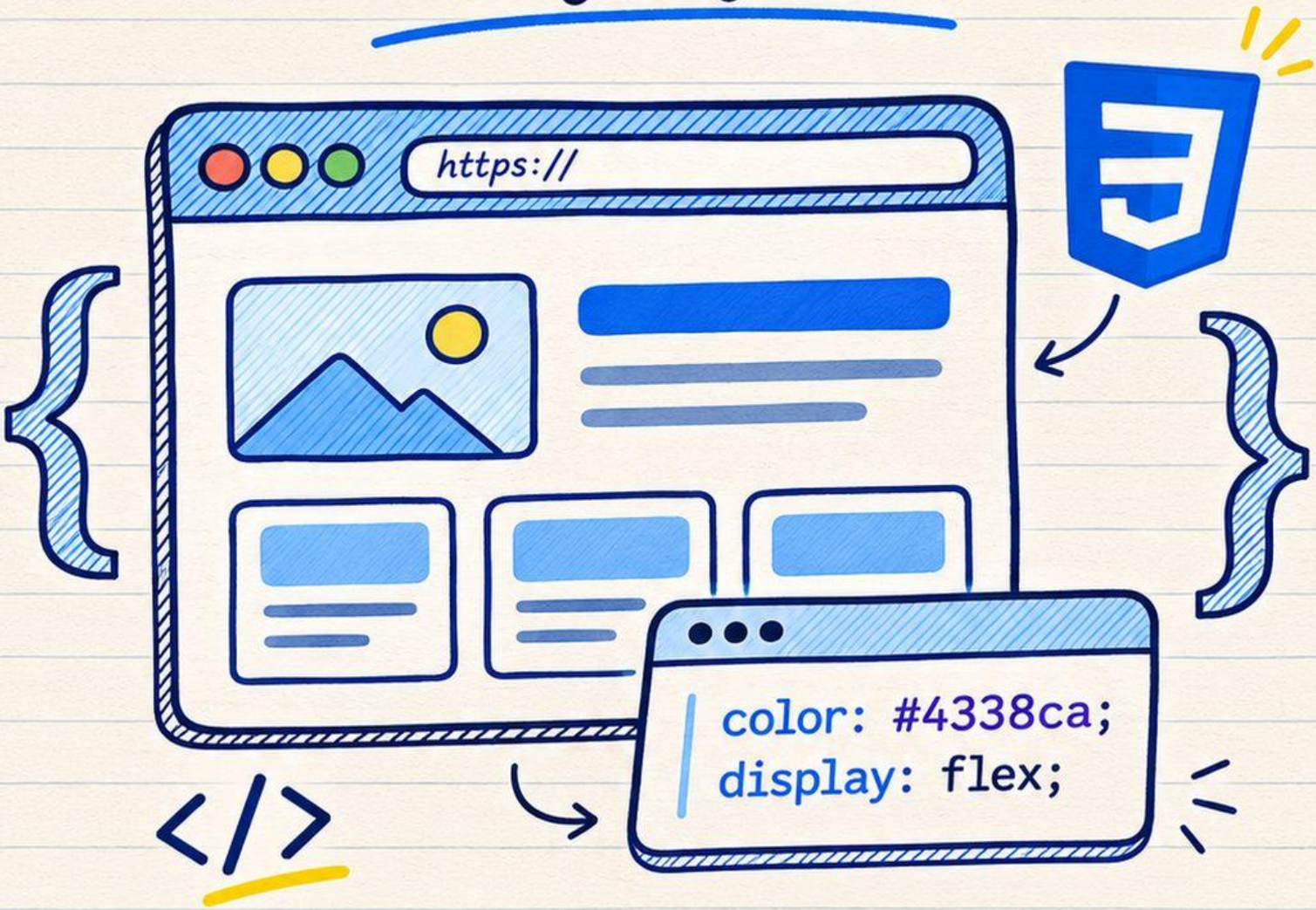
ZainSpire

CSS

HANDWRITTEN NOTES

Cascading Style Sheets



Learn. Practice. Build.

01 / 20 — CSS: THE STARTING POINT

CSS means Cascading Style Sheets.
It controls the presentation of HTML: colour, typography, spacing and layout.

HTML supplies content; CSS styles it.

CONNECT A STYLESHEET

In the HTML head:

```
<link rel="stylesheet" href="style.css">
```

Save your CSS in style.css beside index.html.

FIRST RULE

```
h1 {
    color: #4338ca;
    font-size: 32px;
}
```

h1 is the selector.
Inside braces, each declaration has a property, a colon, a value and a semicolon.

VISUAL RESULT



PRACTICE

Change the heading colour. Save both files and refresh your browser.
CSS comments use /* a comment */.

02/20 — CSS: SELECTORS

SELECTORS: CHOOSE YOUR TARGET

`p` targets paragraph elements.
`.note` targets elements with class="note".
`#intro` targets the element with id="intro".
`*` targets every element.

RELATIONSHIPS

`.card p` selects paragraphs anywhere inside .card.
`.card > p` selects direct-child paragraphs.
`h2 + p` selects the paragraph immediately after h2.
`h1, h2` applies the same rule to both selectors.

EXAMPLE

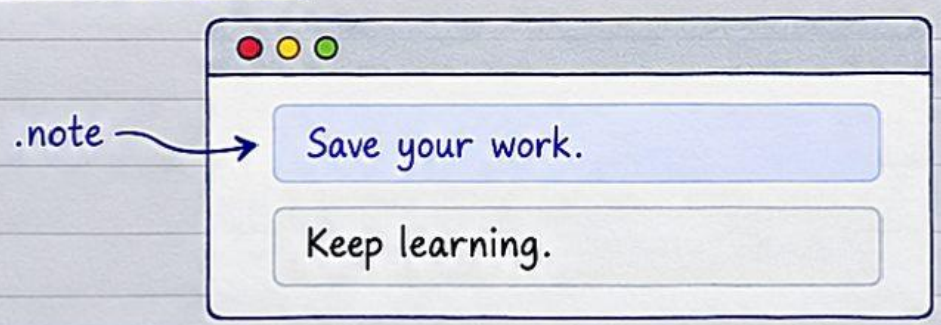
CSS:

```
.note {
    color: #4338ca;
    background: #eef2ff;
}
```

HTML:

```
<p class="note">Save your work.</p>
<p>Keep learning.</p>
```

VISUAL RESULT



REMEMBER

A class can be reused. An id must be unique in its document.
[type="email"] is an attribute selector.

PRACTICE

Style three notes with one class.

03/20 — CSS: THE CASCADE & INHERITANCE

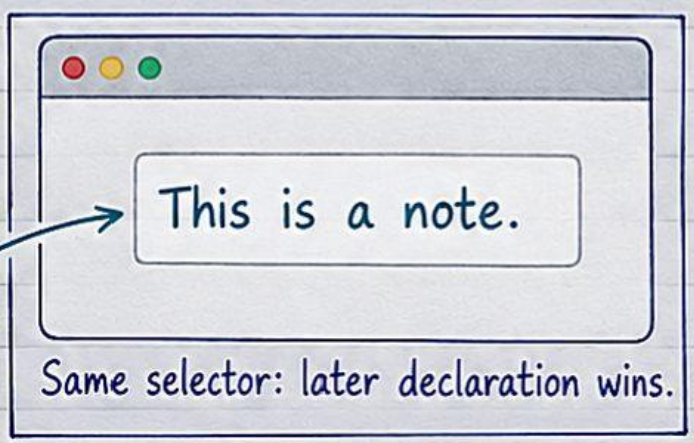
When declarations compete, CSS uses the cascade. Origin, importance and cascade layers are considered before selector specificity. Source order can break a remaining tie.

SPECIFICITY

Within the same origin, importance and layer:
#intro is more specific than .note.
.note is more specific than p.
Compare ID count, then class/attribute/pseudo-class count, then type/pseudo-element count.
The universal selector adds no specificity.

EXAMPLE

```
.note {
  color: purple;
}
.note {
  color: teal;
}
```



INHERITANCE

Text colour and font family commonly inherit from a parent.
Margin and border do not.
inherit requests the parent's computed value.
Avoid using !important as a routine fix.
Inspect matched rules in browser developer tools.

04 / 20 — CSS: THE BOX MODEL

From inside to outside:

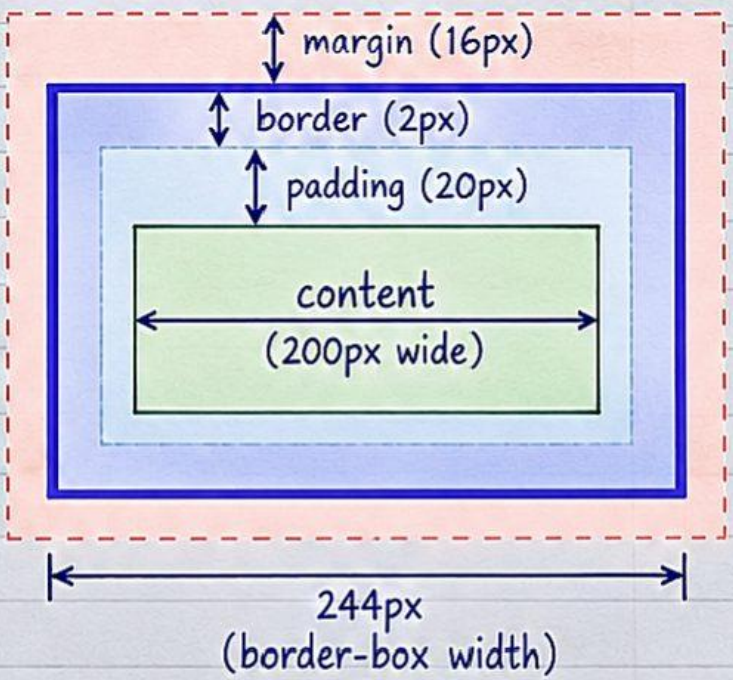
CONTENT → PADDING → BORDER → MARGIN

Padding is inside the border. Margin separates boxes.
Background paints through padding, but not margin.

EXAMPLE

```
.card {
  width: 200px;
  padding: 20px;
  border: 2px solid #4338ca;
  margin: 16px;
}
```

VISUAL DIAGRAM



WIDTH MATH

With the default content-box:

200 + 20 + 20 + 2 + 2 = 244px.

With box-sizing: border-box, the declared width includes padding and border.

Common reset:

```
*, *::before, *::after {
   box-sizing: border-box;
}
```

Vertical block margins can sometimes collapse.

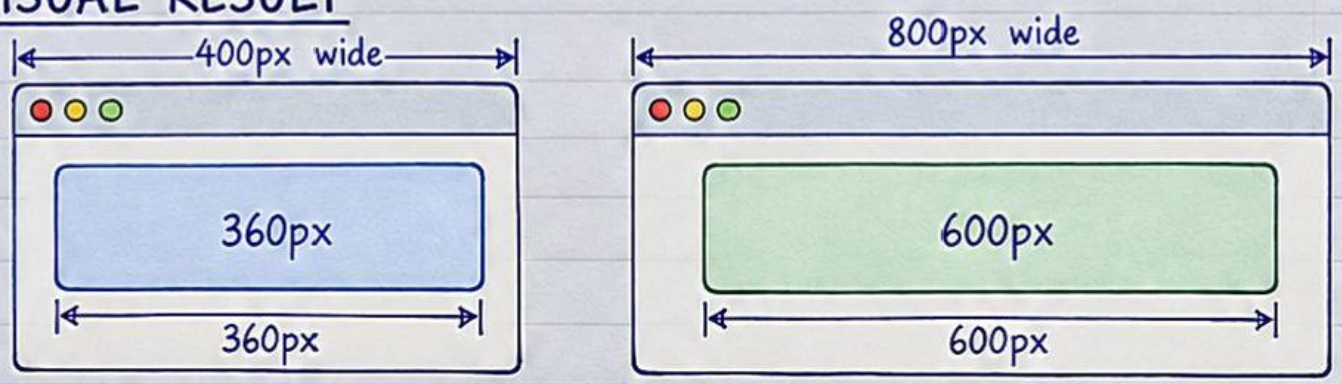
05 / 20 — CSS: UNITS & FLEXIBLE SIZES

px: a CSS pixel.
rem: relative to the root font size.
em: for font-size, relative to the parent's font size.
%: depends on the property and its reference size.
vw: 1% of viewport width.
dvh: 1% of the dynamic viewport height.

EXAMPLE

```
.card {
  box-sizing: border-box;
  width: 90%;
  max-width: 600px;
  padding: 1rem;
}
```

VISUAL RESULT



Note: diagrams illustrate width only; centring is added separately with margin-inline: auto.

USEFUL FUNCTIONS

```
width: min(90%, 600px);
font-size: clamp(1.5rem, 4vw, 3rem);
```

clamp gives a minimum, preferred value and maximum.

PRACTICE

Resize the browser and watch the width change.
Do not assume 1rem is always 16px; users can change their default font size.

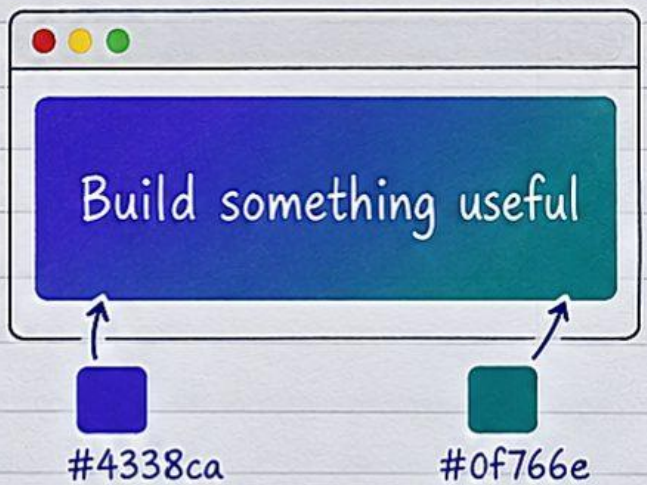
06 / 20 — COLOURS, BACKGROUNDS & GRADIENTS

Colour values include names, hex, rgb() and hsl().
color affects text. background affects the box.

EXAMPLE

```
.banner {
  color: white;
  background:
    linear-gradient(135deg, #4338ca, #0f766e);
  padding: 24px;
}
```

VISUAL RESULT



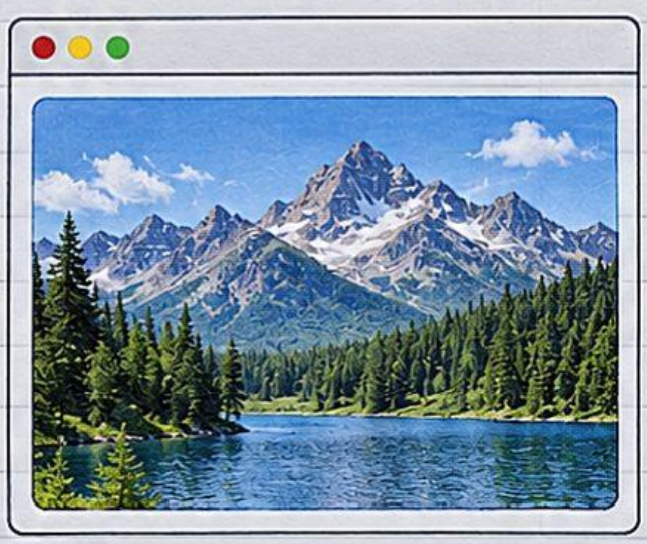
BACKGROUND IMAGES

```
.hero {
  background-image: url("mountains.jpg");
  background-size: cover;
  background-position: center;
}
```

cover fills the box and may crop the image.

Background images have no alt attribute;
use an HTML image for meaningful content.

VISUAL RESULT



REMEMBER

Check text contrast. Do not communicate meaning using colour alone.
opacity fades the entire element, including its children.

07/20 — CSS: TYPOGRAPHY THAT READS WELL

CSS controls the presentation of web pages.
These styles make text easier to read and more pleasant to look at.

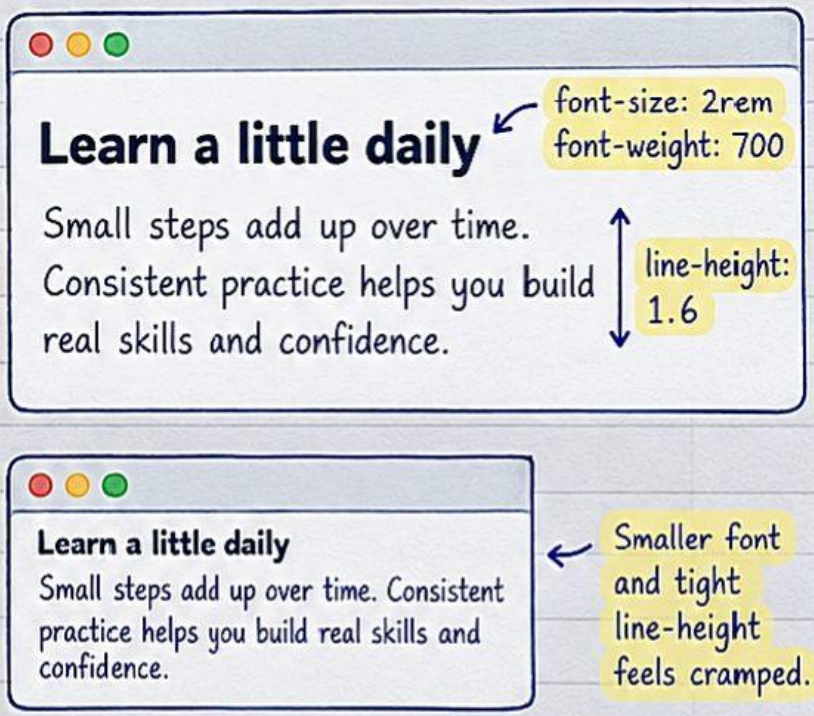
EXAMPLE

```
body {
  font-family: system-ui, sans-serif;
  font-size: 1rem;
  line-height: 1.6;
  color: #1f2937;
}

h1 {
  font-size: 2rem;
  font-weight: 700;
}

.article { max-width: 65ch; }
```

VISUAL RESULT



TEXT TOOLS

`text-align` aligns inline content inside its box.
`text-decoration` can underline links.
`letter-spacing` adjusts space between letters.
`overflow-wrap: anywhere` helps long strings wrap.

GOOD HABITS

Keep body text readable. Use a fallback font.
Unitless line-height scales with the element's font size.
Avoid fixed heights on text boxes: zoom and translation can increase their height.

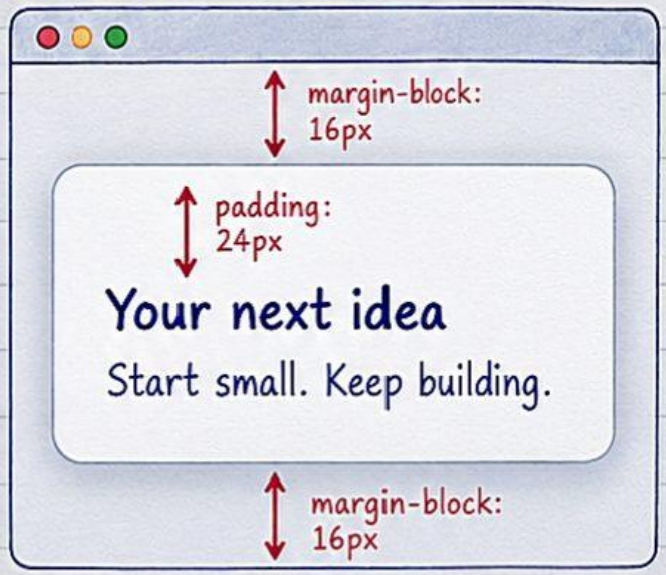
08 / 20 — CSS: SPACING, BORDERS & SHADOWS

CSS controls spacing, borders and shadows to make elements look clear and organised.

EXAMPLE

```
.card {
  padding: 24px;
  margin-block: 16px;
  border: 1px solid #cbd5e1;
  border-radius: 16px;
  box-shadow: 0 6px 18px rgb(0 0 0 / 12%);
}
```

VISUAL RESULT



SHORTHAND ORDER

`padding: 8px 16px;` means top/bottom 8px, left/right 16px.

`padding: 4px 8px 12px 16px;` means top, right, bottom, left.

SHADOW ORDER

Horizontal offset, vertical offset, blur, optional spread, colour.

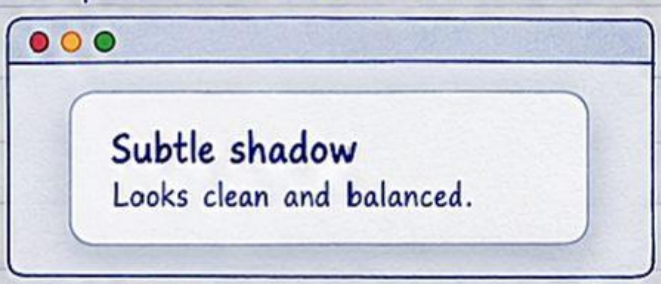
```
box-shadow: 0 6px 18px rgb(0 0 0 / 12%);
```

inset creates an inner shadow.

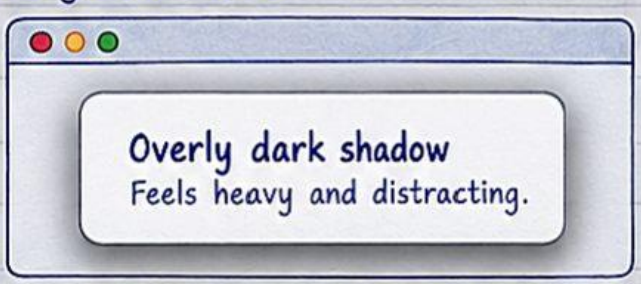
Logical properties such as margin-inline follow writing direction.

PRACTICE

Compare a subtle shadow with an overly dark one.



box-shadow: 0 6px 18px rgb(0 0 0 / 12%);



box-shadow: 0 10px 30px rgb(0 0 0 / 50%);

09/20 — CSS: DISPLAY, FLOW & OVERFLOW

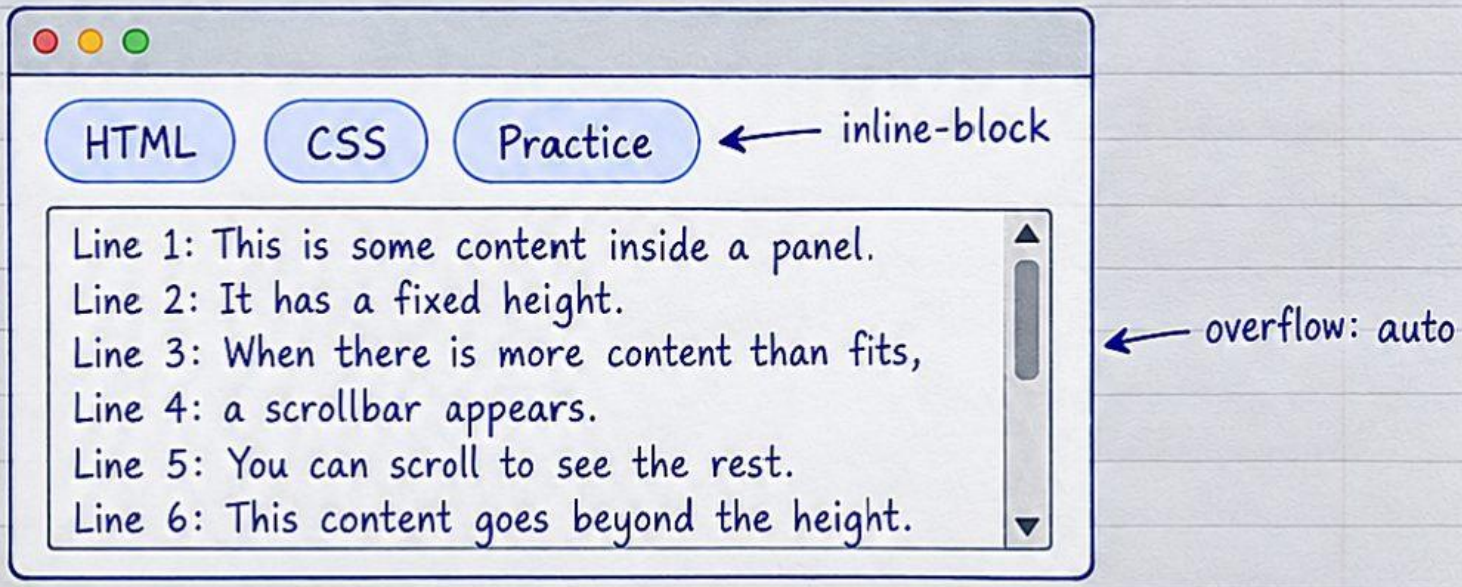
Block boxes normally start on a new line.
Inline boxes flow with surrounding text.
inline-block stays inline but accepts box dimensions.
display: none removes the box from layout.
visibility: hidden hides it while keeping its layout space.

EXAMPLE

```
.chip {
    display: inline-block;
    padding: 6px 12px;
    background: #eef2ff;
    border-radius: 999px;
}
```

```
.panel {
    max-height: 100px;
    overflow: auto;
}
```

VISUAL RESULT



REMEMBER

overflow: hidden clips overflowing content.
Do not hide overflow merely to conceal a broken layout.
A fixed height can cut off translated text.
Use normal document flow as your starting point.

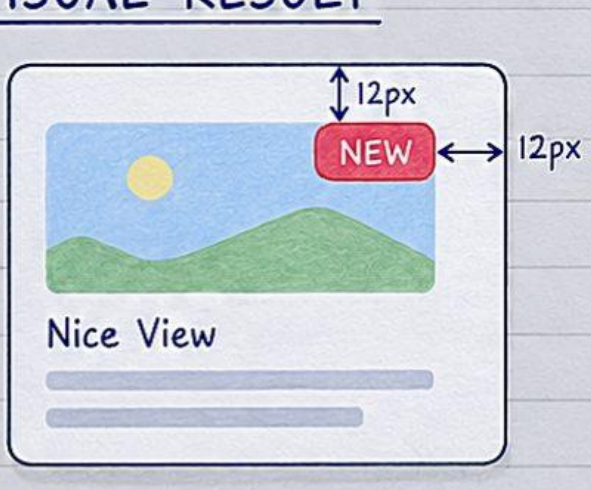
10 / 20 — CSS: POSITIONING & STACKING

static: the normal default.
relative: keeps its flow space and can be offset.
absolute: removed from normal flow; uses a containing block.
fixed: usually positioned against the viewport.
sticky: follows flow, then sticks at a threshold within its scroll context.

EXAMPLE

```
.card {
    position: relative;
}
.badge {
    position: absolute;
    top: 12px;
    right: 12px;
}
.toolbar {
    position: sticky;
    top: 0;
}
```

VISUAL RESULT





Z-INDEX

z-index controls stacking in applicable contexts.
A large z-index cannot escape an ancestor stacking context.
Transforms can create containing blocks and stacking contexts.
Sticky behaviour depends on available space and scrolling ancestors.

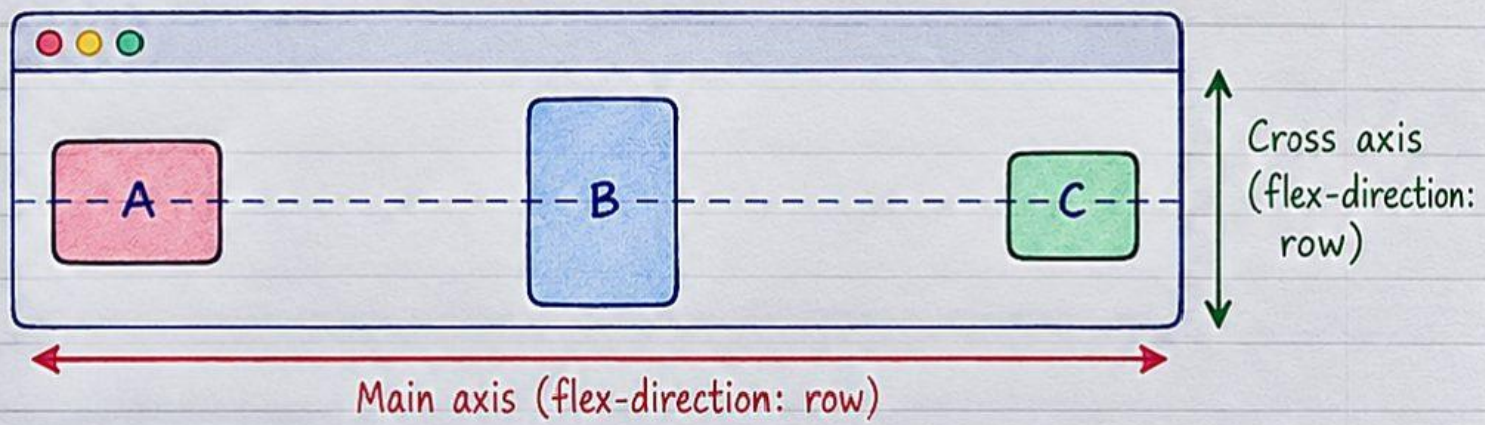
11 / 20 — CSS: FLEXBOX: ARRANGE A ROW OR COLUMN

Use Flexbox for one-dimensional layout.

EXAMPLE

```
.row {
    display: flex;
    gap: 12px;
    justify-content: space-between;
    align-items: center;
    min-height: 100px;
}
```

VISUAL RESULT



KEY CONTROLS

`flex-direction:` row or column.
`justify-content` distributes along the main axis.
`align-items` aligns items on the cross axis.
`gap` creates space between items.
`flex-wrap:` wrap allows extra lines.

REMEMBER

Changing direction changes the axes.
space-between needs free space to distribute.
The direct children become flex items.

PRACTICE

PRACTICE: change row to column and compare.

12 / 20 — CSS: FLEX ITEMS & WRAPPING

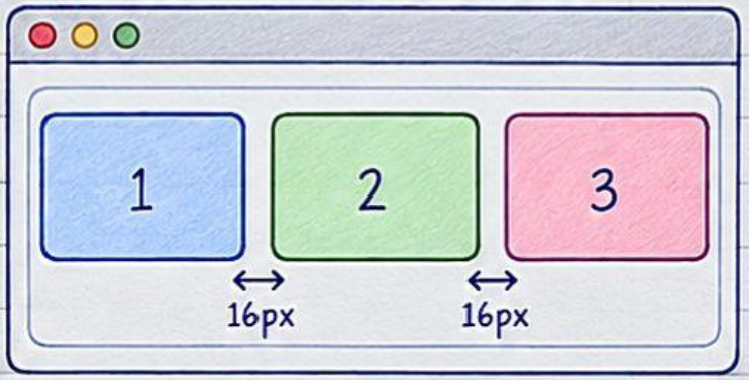
EXAMPLE

```
.cards {
    display: flex;
    flex-wrap: wrap;
    gap: 16px;
}

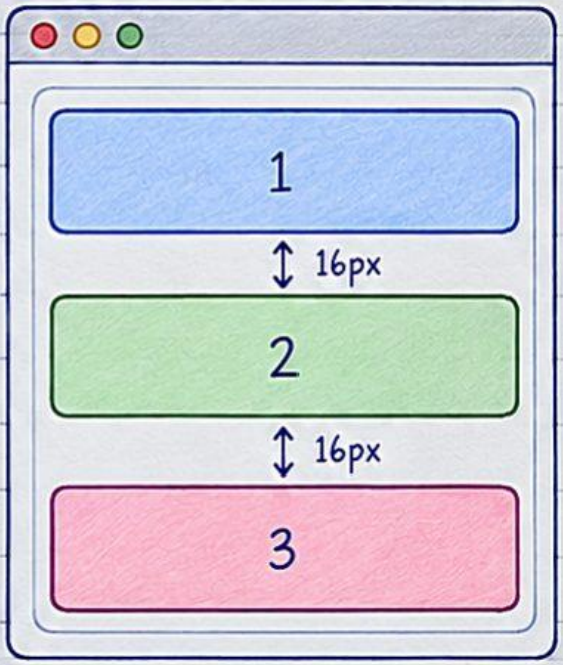
.card {
    flex: 1 1 220px;
    min-width: 0;
}
```

VISUAL RESULT

Wide container (cards in a row)



Narrow container (cards stacked)



READ THE SHORTHAND

1 = flex-grow: share positive free space.
1 = flex-shrink: allow shrinking when needed.
220px = flex-basis: the starting main size.

OTHER TOOLS

align-self overrides alignment for one item.
margin-left: auto can push an item away in a row.
min-width: 0 can let a flex item shrink below its intrinsic minimum width.

ACCESSIBILITY

order changes visual order, not document or keyboard order.
Keep the source order meaningful.

PRACTICE

Add a fourth card and resize.

13 / 20 — CSS: GRID ROWS AND COLUMNS

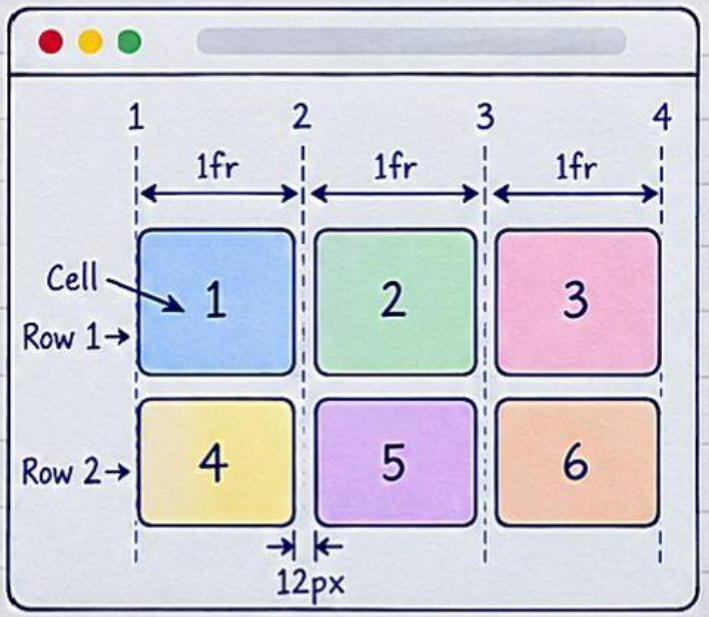
`GRID` is useful for two-dimensional layouts.

EXAMPLE

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 12px;
}

.item { min-width: 0; }
```

VISUAL RESULT



TERMS

`Track:` a row or column.

`Cell:` one row-column intersection.

`Area:` one or more cells.

`fr:` shares available space among flexible tracks.

SIZING

`repeat(3, 1fr)` repeats a track three times.

`minmax(0, 1fr)` allows a flexible track to shrink below an automatic content minimum.

Long content can overflow: test real text.

PRACTICE

Change three columns to two.

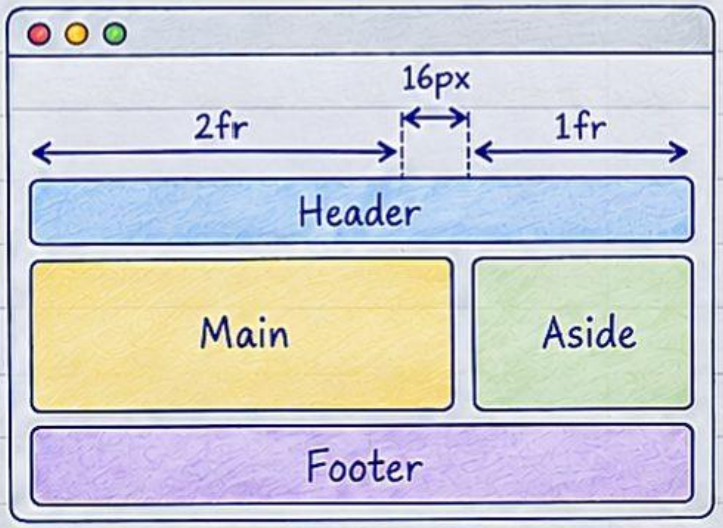
14 / 20 — CSS: GRID PLACEMENT & AUTO-FIT

EXAMPLE

```
.layout {
  display: grid;
  grid-template-columns: 2fr 1fr;
  gap: 16px;
}

.header { grid-column: 1 / -1; }
.footer { grid-column: 1 / -1; }
```

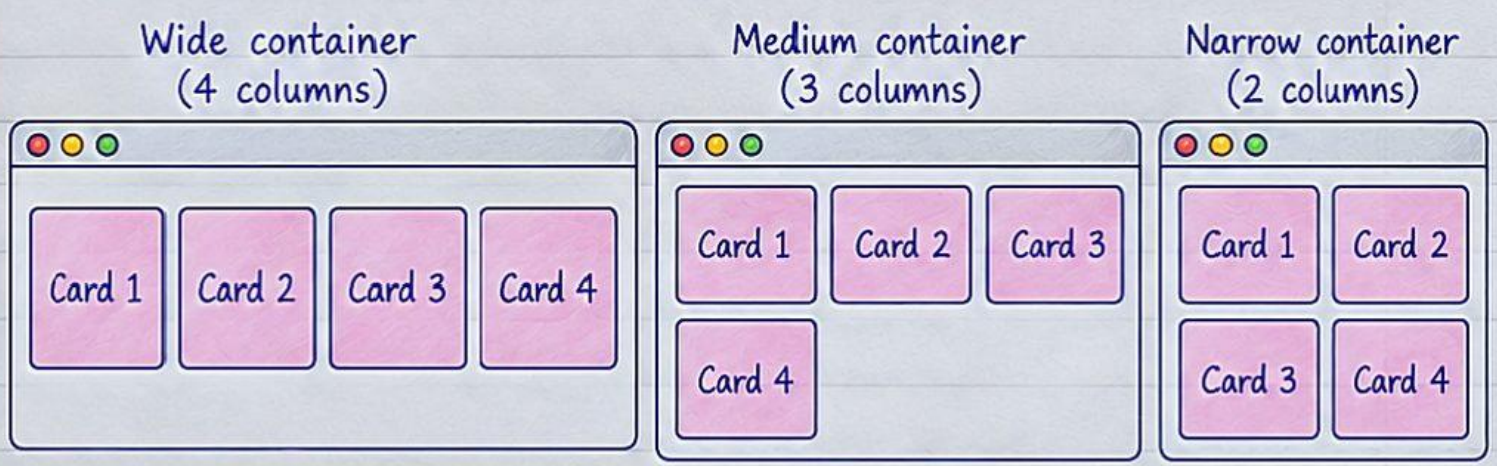
VISUAL RESULT



AUTOMATIC CARD COLUMNS

```
.cards {
  display: grid;
  gap: 16px;
  grid-template-columns:
    repeat(auto-fit, minmax(min(100%, 220px), 1fr));
}
```

The minimum adapts to very narrow containers.
Available width controls how many columns fit.



REMEMBER

grid-column: 1 / -1 spans the explicit columns.
auto-fit collapses unused tracks.
Keep source order logical even when changing visual placement.

15 / 20 — CSS: RESPONSIVE DESIGN & MEDIA QUERIES

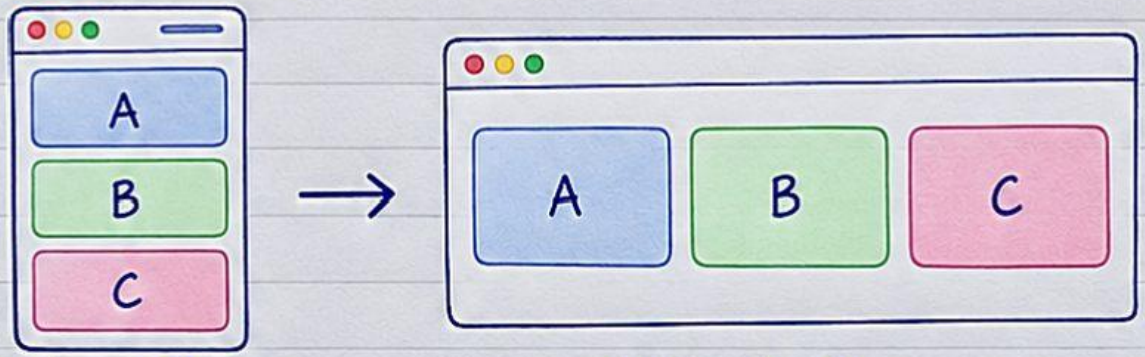
Start with a narrow layout. Add changes when content needs more room.

EXAMPLE

```
.cards {
    display: grid;
    gap: 16px;
    grid-template-columns: 1fr;
}

@media (min-width: 48rem) {
    .cards {
        grid-template-columns: repeat(3, 1fr);
    }
}
```

VISUAL RESULT



Below 48rem

At least 48rem

FOUNDATIONS

- Include the viewport meta tag in the HTML head.
- img { max-width: 100%; height: auto; }
- Use flexible widths and readable text.
- Test long words, zoom and different content lengths.

PRACTICE

- Resize slowly and find where the layout becomes cramped.
- Choose breakpoints for your content, not just named devices.
- A small screen does not always mean touch input.

16 / 20 — CSS: STATES, PSEUDO-ELEMENTS & FORMS

EXAMPLE

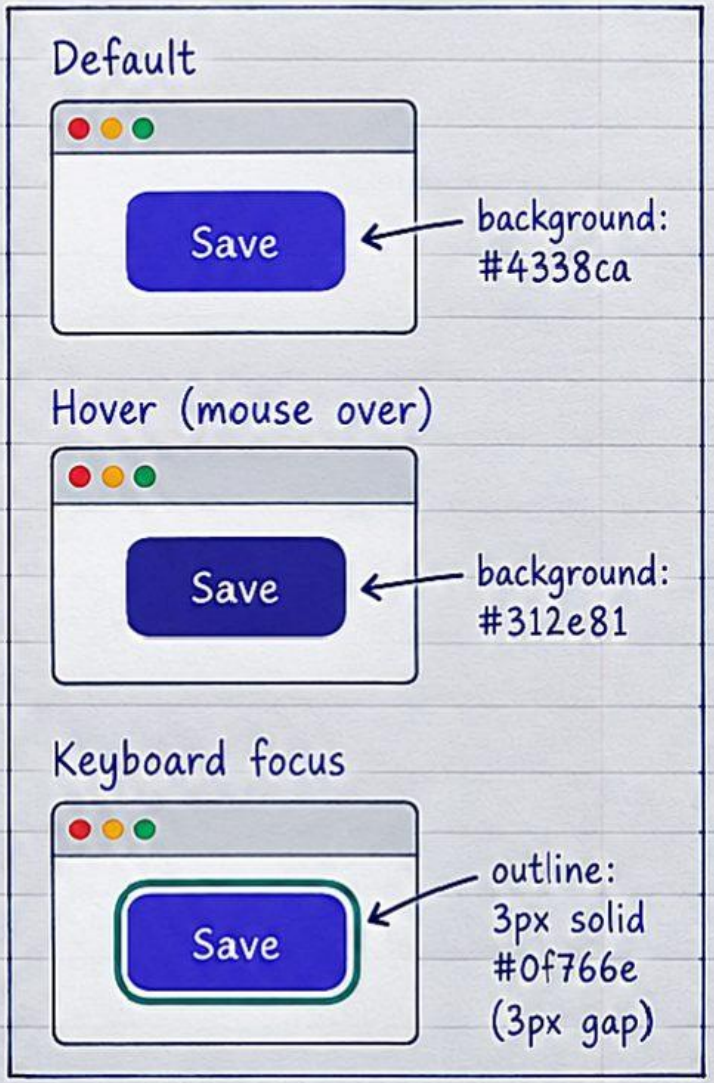
```
button {
  padding: 10px 18px;
  background: #4338ca;
  color: white;
  border: 0;
  border-radius: 8px;
}

button:hover {
  background: #312e81;
}

button:focus-visible {
  outline: 3px solid #0f766e;
  outline-offset: 3px;
}

input, button { font: inherit; }
```

VISUAL RESULT



MORE SELECTORS

`:checked` matches checked controls.
`:disabled` matches disabled controls.
`:first-child` selects a first child.
`::before` and `::after` generate pseudo-elements. They normally need content to appear.

REMEMBER

Keep real labels in HTML.
Do not put essential instructions only in generated content.
Hover cannot be the only way to reveal important information.

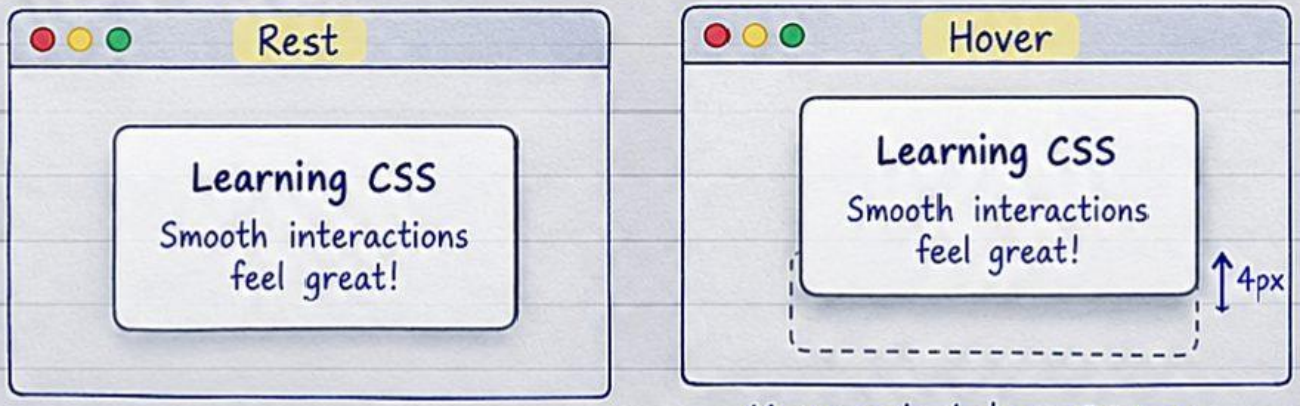
17 / 20 — CSS: TRANSFORMS & TRANSITIONS

A transform changes visual appearance without rearranging normal flow.

EXAMPLE

```
.card {
  transition: transform 200ms ease;
}
.card:hover {
  transform: translateY(-4px);
}
```

VISUAL RESULT



Movement takes 200ms.

TRANSFORM TOOLS

translate() moves.
rotate() turns.
scale() resizes visually.
transform-origin sets the pivot.
Order matters when combining transforms.

REDUCED MOTION

```
@media (prefers-reduced-motion: reduce) {
  .card { transition: none; transform: none; }
  .card:hover { transform: none; }
}
```

Transition specific properties rather than all.
Use motion to explain a change; avoid distracting movement.

PRACTICE

try a gentle colour transition on a button.

18 / 20 — CSS: KEYFRAME ANIMATIONS

Keyframes describe values at different moments.

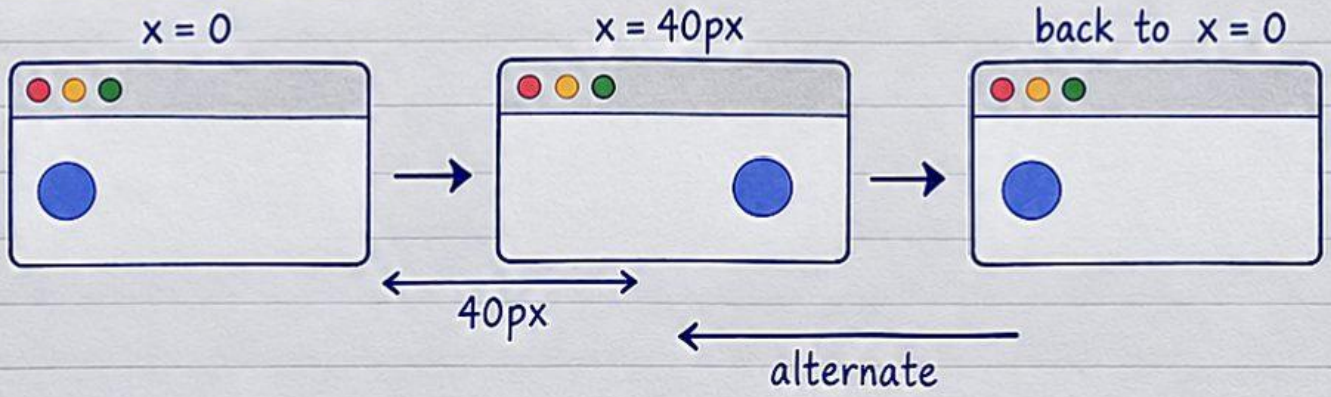
EXAMPLE

```
.dot {
    animation: move 1s ease-in-out infinite alternate;
}

@keyframes move {
     from { transform: translateX(0); }
     to   { transform: translateX(40px); }
}

@media (prefers-reduced-motion: reduce) {
    .dot { animation: none; }
}
```

VISUAL RESULT



ANIMATION PARTS

Name, duration, timing, delay, iteration count, direction and fill mode.
A transition reacts to a value change.
An animation can run through defined keyframes.

GOOD HABITS

Provide pause controls for persistent nonessential motion.
Avoid flashing effects.
Use transform and opacity when they suit the effect, and test performance.

19 / 20 — CSS: CUSTOM PROPERTIES & REUSABLE THEMES

CSS custom properties let you store values in variables. You can change the values and reuse the same rules to create different themes.

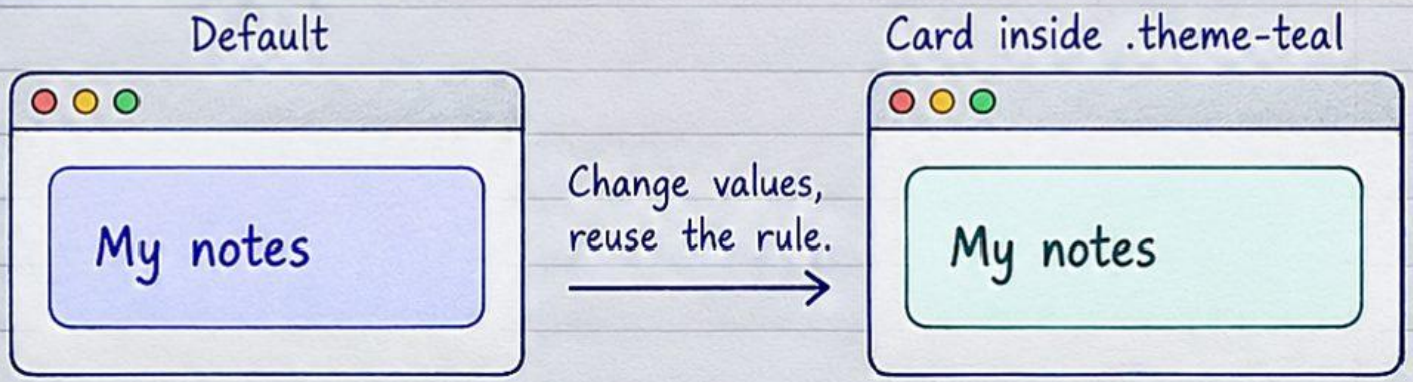
EXAMPLE

```
:root {
  --accent: #4338ca;
  --surface: #eef2ff;
  --space: 16px
}
```

```
.card {
  color: var(--accent);
  background: var(--surface);
  padding: var(--space);
}
```

```
.theme-teal {
  --accent: #0f766e;
  --surface: #f0fdfa;
}
```

VISUAL RESULT



HOW THEY WORK

Custom properties participate in the cascade and normally inherit.
var(--accent, purple) supplies a fallback when the variable cannot provide a usable value.
Names are case-sensitive.
A value can be invalid for the property that receives it.

PRACTICE

Define shared colours, spacing and corner radius.
Change one variable and compare all affected components.

20 / 20 — CSS: MINI PROJECT & FINAL CHECK

BUILD A RESPONSIVE RESOURCE LIBRARY

Use a page heading, a short introduction and a .cards container with three .card articles in HTML.
Connect your stylesheet and build on this CSS:

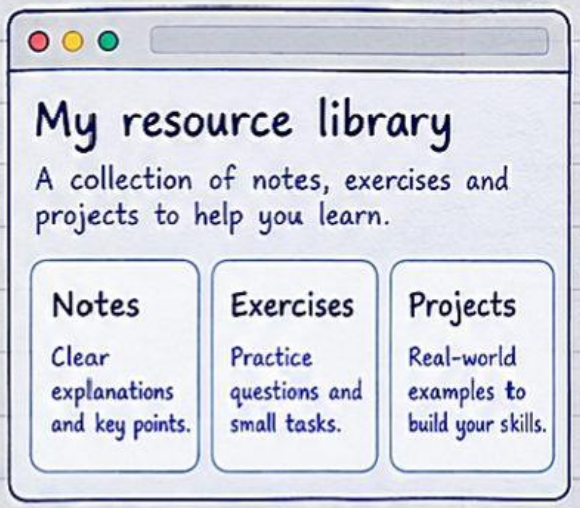
HTML (index.html)

```
<h1>My resource library</h1>
<p>A collection of notes, exercises and projects
 to help you learn.</p>
<div class="cards">
  <article class="card">
    <h2>Notes</h2>
    <p>Clear explanations and key points.</p>
  </article>
  <article class="card">
    <h2>Exercises</h2>
    <p> Practice questions and small tasks.</p>
  </article>
  <article class="card">
    <h2>Projects</h2>
    <p> Real-world examples to build your skills.</p>
  </article>
</div>
```

CSS (style.css)

```
* { box-sizing: border-box; }
body {
      margin: 0;
      padding: 24px;
      font-family: system-ui, sans-serif;
      background: #f8fafc;
      color: #1f2937;
}
.cards {
      display: grid;
      gap: 16px;
      grid-template-columns:
          repeat(auto-fit, minmax(min(100%, 220px), 1fr));
}
.card {
       padding: 20px;
       background: white;
       border: 1px solid #cbd5e1;
}      border-radius: 12px;
```

VISUAL RESULT



Wide screen (cards in a row)



Narrow screen (cards stacked)

FINAL CHECK

Test keyboard focus, contrast, zoom, narrow screens and long text.
Inspect crossed-out rules and box sizes in developer tools.
Check missing files and invalid property values.
Add your own content and practise one improvement at a time.