🧾 Milestone-2 Cheatsheet

📏 CSS Measuring Units

Common Units:

Unit	Description	Use Case
px	Absolute pixel size	Borders, fixed icons
%	Percentage of parent	Width/height relative sizing
em	Relative to parent font size	Padding/margins
rem	Relative to root font size	Consistent spacing
vw	% of viewport width	Full-screen sections
vh	% of viewport height	Hero banners, full-height

Tips:

- Use %, em, rem for responsive layout
- Use min-width for mobile-first media queries

🧵 Media Query for Basic Responsiveness

Syntax:

```
@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }
}
```

Key Breakpoints (Common):

- Mobile: max-width: 600px
- Tablet: max-width: 768px
- Desktop: min-width: 1024px

📱 7-7 Responsive Queries for All Devices

Device-specific Queries:

```
/* Small Devices */
@media (max-width: 480px) { }

/* Medium Devices */
@media (min-width: 481px) and (max-width: 768px) { }

/* Large Devices */
@media (min-width: 769px) { }
```

Best Practice: Design mobile-first and expand upward.

✅ Seven Must-Do Things for Responsive Websites

1. **Use Fluid Layouts** (%, vw, em)
2. **Set Viewport Meta Tag**

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

3. **Flexible Images (max-width: 100%)**
4. **Media Queries for Layout Adjustment**
5. **Use Flex/Grid for structure**
6. **Avoid fixed widths & heights**
7. **Test on real devices & emulators**

🎯 Flexbox & CSS Grid

🧱 FLEXBOX — One-Dimensional Layout System

✅ Activate Flexbox

```
.container {
  display: flex;
}
```

🔀 Main Axis Control (Horizontal by default)

Property	Values	Description
flex-direction	row (default), column, row-reverse, column-reverse	Defines direction of items
justify-content	flex-start, center, space-between, space-around, space-evenly, flex-end	Aligns items **horizontally** along main axis
gap / column-gap / row-gap	10px, 1rem etc.	Adds space between flex items

---

🎯 Cross Axis Control (Vertical by default)

Property	Values	Description
align-items	flex-start, center, stretch, flex-end, baseline	Aligns items **vertically** in the container
align-content	Like align-items, but works when multiple rows	

🧩 Item-Level Properties

Property	Values	Description
flex-grow	0, 1 (default: 0)	Controls how much an item grows
flex-shrink	0, 1	Controls shrinking of items
flex-basis	auto, 200px, 50%	Sets initial size before growing/shrinking
flex	shorthand (flex-grow flex-shrink flex-basis)	e.g. flex: 1 1 200px

align-self	Overrides align-items for a single item	

📦 Flexbox Responsive Tips

- Use flex-wrap: wrap to wrap items on small screens
- Use media queries to change flex-direction from row to column

```
@media (max-width: 768px) {
 .container {
  flex-direction: column;
 }
}
```

🎯 CSS GRID — Two-Dimensional Layout System

✅ Activate Grid

```
.container {
 display: grid;
}
```

🏗️ Grid Structure Basics

Property	Values	Description
grid-template-columns	200px 200px, 1fr 2fr, repeat(3, 1fr)	Defines number & width of columns
grid-template-rows	100px auto, repeat(2, 1fr)	Defines number & height of rows
gap	20px, row-gap, column-gap	Sets spacing between rows/columns
grid-auto-rows	min-content, auto, 200px	Sets height for rows not defined
grid-auto-columns	Like above, but for columns	

🧭 Aligning in Grid

Property	Values	Description

justify-items	start, center, end, stretch	Aligns grid items horizontally inside their cell
align-items	start, center, end, stretch	Aligns items vertically inside their cell
justify-content	For the entire grid container alignment (horizontal)	
align-content	For the entire grid container alignment (vertical)	

🧩 Grid Item Placement

Property	Description
grid-column-start, grid-column-end	Define where item starts/ends on columns
grid-row-start, grid-row-end	Same, but for rows
grid-column: 1 / 3;	Span from column 1 to 3

grid-row: 2 / span 2;	Start at row 2, span 2 rows

Shorthand:

```
.item {
  grid-column: 1 / span 2;
  grid-row: 2 / 4;
}
```

🗺️ Named Grid Areas

```
.container {
  display: grid;
  grid-template-areas:
    "header header"
    "sidebar content"
    "footer footer";
}


.header {
  grid-area: header;
}
```

💡 COMPARING FLEXBOX vs GRID

Feature	Flexbox	Grid
Axis	One (row/column)	Two (row + column)
Control	Content-based	Layout-based
Best for	Navbars, Forms	Page layouts, cards, dashboards
Item placement	Sequential	Precise placement

📱 RESPONSIVE DESIGN TIPS (Flex + Grid)

1. **Media Queries**

```
@media (max-width: 768px) {
 .container {
```

```
    flex-direction: column;
    grid-template-columns: 1fr;
  }
}
```

2. **Use minmax() in Grid**

```
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
```

3. **Switch between Flex/Grid based on screen size**
   Use Flexbox for simple row/column, Grid for layout control.

🧩Component-Based vs Utility-Based Design

🧱 Component-Based Design (Traditional)

- **Approach**: Write your own CSS classes in separate `.css` files.
- **Advantages**:
  - Reusable
  - Clean HTML
- **Disadvantages**:

- Slow iteration
- Class bloat

🔧 **Example**:

```
.card {
  background-color: white;
  padding: 1rem;
  border-radius: 0.5rem;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}


<div class="card">...</div>
```

⚡ Utility-Based Design (Tailwind Style)

- **Approach**: Apply pre-built classes directly in HTML
- **Advantages**:
  - Rapid development
  - Predictable, atomic

- Responsive & mobile-first

- **Disadvantages**:
  - Verbose HTML (unless using components like in React or Blade)

🧪 **Example**:

...

🧪 Utility/Class Based Design Technique Example

A **blog card example** with full Tailwind classes:

  <h2 class="text-2xl font-semibold text-gray-800 mb-2">Post Title</h2>
  Post description goes here.

🖋️ Tailwind Typography, Background, Text & Border

📌 Typography

Class	Purpose

`text-xs` → `text-9xl`	Font sizes
`font-light` → `font-black`	Font weights
`text-center`, `text-left`	Alignment
`tracking-wider`, `leading-loose`	Letter/line spacing
`uppercase`, `capitalize`	Text case

<h1 class="text-3xl font-bold text-center uppercase">Hello</h1>

🎨 Backgrounds

Class	Description
`bg-red-100` to `bg-red-900`	Background colors
`bg-gradient-to-r from-indigo-500 to-pink-500`	Gradients

✍️ Borders

Class	Description
`border`, `border-2`, `border-t`	Sides and thickness
`border-gray-300`	Color
`rounded`, `rounded-lg`, `rounded-full`	Radius styles

📦 Tailwind Spacing & Box Model

Margin & Padding

- `m`, `mx`, `my`, `mt`, `mb`, `ml`, `mr`
- `p`, `px`, `py`, etc.
- Numeric scale: `0` → `96` (spacing unit = 0.25rem)

Content

Width/Height

Class	Example
`w-full`, `w-1/2`, `w-96`	Width
`h-64`, `min-h-screen`	Height

🕶️Tailwind Flex Layout + Responsive Flex

Item 1

Item 2

🧱 Flex Utilities:

Class	Description
`flex`, `inline-flex`	Display
`flex-row`, `flex-col`	Direction

`justify-*`	Main axis
`items-*`	Cross axis
`gap-*`	Item spacing
`flex-wrap`, `basis-1/2`, `grow`, `shrink`	Advanced control

🔲 Tailwind Grid Layout + Responsive Grid

  Wide

  Box

🌐 Grid Utilities:

Class	Description
`grid-cols-*`	Columns

`grid-rows-*`	Rows
`col-span-*`, `row-span-*`	Item span
`gap`, `place-items`, `justify-items`	Spacing/alignment

🔁 Responsive Grid Layout:

grid-cols-1 sm:grid-cols-2 md:grid-cols-3

🎯State-Based Classes (`hover`, `focus`, `active`, `peer`)

Prefix	Description
`hover:`	On mouse hover
`focus:`	On input focus
`active:`	On button click
`disabled:`	When disabled

`group-hover:`	Parent element hover
`peer-focus:`	Related sibling focus

<input class="peer ...">

Focus me!

🔁 Dynamic classes:

<button class="bg-blue-500 hover:bg-blue-700 focus:ring-2 focus:ring-blue-400">Click</button>

✅ Summary Points:

- Utility-first = Tailwind's strength
- Design faster with spacing, layout, and state classes
- Responsive design is built-in
- Combines well with component frameworks (React, Vue, Blade)

📽️Breakpoint Prefixes:

Breakpoint	Prefix	Min Width
`sm`	`sm:`	640px
`md`	`md:`	768px
`lg`	`lg:`	1024px
`xl`	`xl:`	1280px
`2xl`	`2xl:`	1536px

Responsive Class Structure:

Text

...

✅ **Mobile-first**: Define base styles, override as the screen gets wider.
