Tailwind CSS Cheatsheet

A comprehensive cheat sheet for Tailwind CSS, covering essential utilities, directives, and concepts for rapid UI development.

Core Concepts & Setup

Installation

Using npm:

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

Using yarn:

```
yarn add -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

Configure your template paths in `tailwind.config.js`:

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./src/**/*.{html,js}"
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Add Tailwind directives to your CSS:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Utility-First Fundamentals

Tailwind CSS is a utility-first CSS framework. Instead of pre-designed components, it provides low-level utility classes that you compose to build custom designs directly in your HTML.

Example:

```
<button class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded">
  Click me
</button>
```

This creates a blue button that changes color on hover, has white text, bold font, padding, and rounded corners.

Common Utilities

Layout

`container`	Centers content horizontally with maximum width based on breakpoints.
`block`, `inline`, `inline-block`	Sets the display property of an element.
`grid`, `flex`	Enables CSS Grid and Flexbox layouts.
`static`, `relative`, `absolute`, `fixed`, `sticky`	Sets the position property of an element.

Spacing

`p-*` (padding)	Sets padding on all sides of an element. E.g., `p-4`.
`pt-*`, `pb-*`, `pl-*`, `pr-*`	Sets padding on the top, bottom, left, and right sides respectively. E.g., `pt-2`.
`m-*` (margin)	Sets margin on all sides of an element. E.g., `m-8`.
`mt-*`, `mb-*`, `ml-*`, `mr-*`	Sets margin on the top, bottom, left, and right sides respectively. E.g., `mb-4`.
`space-x-*`, `space-y-*`	Adds space between elements in a flex or grid container. E.g., `space-x-2`.

Typography

`font-*`	Sets the font family. E.g., `font-sans`, `font-serif`, `font-mono`.
`text-*`	Sets the text size. E.g., `text-sm`, `text-lg`, `text-xl`.
`font-bold`, `font-medium`, `font-light`	Sets the font weight.
`italic`, `not-italic`	Sets the font style.
`underline`, `line-through`, `no-underline`	Sets text decoration.
`text-left`, `text-center`, `text-right`, `text-justify`	Sets text alignment.

Customization & Directives

Configuration

The `tailwind.config.js` file allows you to customize Tailwind's default configuration, including colors, fonts, breakpoints, and more.

Example: Extending the theme

```
module.exports = {
  theme: {
    extend: {
      colors: {
        'custom-blue': '#123456',
      },
      fontFamily: {
        'brand': ['Roboto', 'sans-serif'],
      },
    },
  },
}
```

This adds a custom color `custom-blue` and a custom font family `brand`.

Directives

`@tailwind`	Used to inject Tailwind's base, components, and utilities styles.
`@apply`	Allows you to use Tailwind's utility classes directly in your own CSS. Useful for extracting reusable component styles. `.btn { @apply bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded; }`
`@layer`	Used to organize custom CSS into layers (base, components, utilities). This helps Tailwind properly optimize and order the CSS. `@layer components { .btn { @apply bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded; } }`
`@config`	Specifies a custom Tailwind configuration file. `@config 'path/to/tailwind.config.js';`

Responsive Design & State Variants

Responsive Modifiers

Tailwind uses breakpoint prefixes to apply styles conditionally based on screen size.
`sm:`, `md:`, `lg:`, `xl:`, `2xl:`

Example:

```
<div class="text-center md:text-left">
  This text will be centered on small screens and left-aligned on medium screens and up.
</div>
```

State Variants

`hover:`	Applies styles on hover.
`focus:`	Applies styles when the element is focused.
`active:`	Applies styles when the element is active (e.g., being clicked).
`disabled:`	Applies styles when the element is disabled.
`focus-within:`	Applies styles to the parent when a child element is focused.
`focus-visible:`	Applies styles when the element is focused using the keyboard (accessibility).

Dark Mode

Tailwind supports dark mode using the `dark:` variant. You need to configure dark mode in your `tailwind.config.js` file.

```
// tailwind.config.js
module.exports = {
  darkMode: 'class', // or 'media'
  // ...
}
```

Then, you can use the `dark:` prefix to apply styles in dark mode:

```
<div class="bg-white dark:bg-gray-800 text-gray-800 dark:text-white">
  This text will be dark on light backgrounds and light on dark backgrounds.
</div>
```