Tailwind CSS

Docs = https://tailwindcss.com/docs/installation

Playground for Tailwind CSS = https://play.tailwindcss.com/

Ready-to-use Tailwind CSS blocks = https://tailblocks.cc/

Tailwind CSS is a highly customizable, utility-first CSS framework that allows developers to build responsive designs directly in their HTML. It differs from traditional CSS frameworks like Bootstrap in that it doesn't provide pre-designed components; instead, it offers low-level utility classes that let you build custom designs without writing CSS.

Key Features of Tailwind CSS

1. **Utility-First Approach**:
    - Tailwind CSS provides utility classes for almost everything: margins, padding, colors, font sizes, shadows, and more.
    - Each class represents a single CSS property, so instead of creating custom CSS rules, you apply classes directly to elements.

    For example:

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

    - Here, `bg-blue-500` applies a blue background color, `text-white` sets the text color to white, `font-bold` makes the text bold, `py-2` adds vertical padding, `px-4` adds horizontal padding, and `rounded` applies rounded corners.
2. **Responsive Design**:
    - Tailwind includes responsive utility classes that you can apply based on breakpoints.
    - You can use `sm:`, `md:`, `lg:`, `xl:` and `2xl:` prefixes to make your designs responsive.

For example:

```
<div class="text-sm md:text-lg lg:text-xl">
    Responsive Text Size
</div>
```

- Here, the text size changes based on the screen size: `text-sm` on small screens, `md:text-lg` on medium screens, and `lg:text-xl` on large screens.

3. **Highly Customizable**:
    - Tailwind allows you to customize its default theme via a configuration file (`tailwind.config.js`). You can extend colors, spacing, fonts, breakpoints, and more to match your design needs.
    - Tailwind's configuration file enables you to create a custom design system for your projects.
4. **Dark Mode and Variants**:
    - Tailwind provides built-in support for dark mode. You can create dark-mode variants by adding the `dark:` prefix.
    - Variants can be extended for states like hover, focus, active, etc.

    For example:

```
<div class="bg-white dark:bg-gray-800">
    Dark Mode Background
</div>
```

5. **CSS Purge**:
    - Tailwind automatically removes unused CSS classes when building your project for production, resulting in smaller CSS files.
6. **Component Composition**:

- Although Tailwind is utility-first, you can still create reusable components by combining utility classes into groups using your own naming conventions. This is often done with tools like React or with template partials.

For example:

```
<div class="card bg-white shadow-lg rounded-lg p-6">
    <!-- Card content goes here -->
</div>
```

7. **No Opinionated Styles**:
    - Unlike Bootstrap or Materialize, Tailwind doesn't provide pre-styled components (like buttons or navbars). This gives you complete freedom over the design without having to override unwanted default styles.

Benefits of Tailwind CSS

- **Rapid Prototyping**: Tailwind allows developers to quickly prototype and build responsive layouts without switching between HTML and CSS files.
- **No Custom CSS Needed**: By using utility classes, you avoid the need to write custom CSS, which can reduce CSS file sizes and simplify maintenance.
- **Design Consistency**: Using predefined classes enforces design consistency across your project since you're using a consistent set of spacing, colors, fonts, etc.
- **Responsive Design Made Easy**: The responsive classes allow for quick adaptation of layouts based on screen sizes.

Why Use Tailwind CSS?

- **Customization**: Tailwind allows you to customize every aspect of your project, making it easy to adapt to any design.
- **Simplicity**: Developers don't need to memorize complex CSS rules. By combining utility classes, you can create complex designs with simple building blocks.
- **Efficiency**: Tailwind's utility classes help avoid writing repetitive CSS and improve development speed.

Example with Tailwind

Below is a basic example of using Tailwind classes to build a card:

```
<div class="max-w-sm mx-auto bg-white shadow-md rounded-lg overflow-hidden">
  <img class="w-full h-48 object-cover" src="image.jpg" alt="Card Image">
  <div class="p-4">
    <h2 class="text-xl font-bold">Card Title</h2>
    <p class="text-gray-600 mt-2">This is a simple card component built with Tailwind CSS.</p>
    <button class="mt-4 bg-blue-500 text-white py-2 px-4 rounded hover:bg-blue-700">
      Learn More
    </button>
  </div>
</div>
```

Summary

Tailwind CSS is a utility-first CSS framework designed for rapid UI development. It allows you to build responsive, customizable layouts without the need for custom CSS, resulting in quicker and more consistent development. Tailwind is highly flexible, making it ideal for modern web development.

Tailwind CSS Setup

Here's a step-by-step guide to setting up **Tailwind CSS** in a basic HTML project. Tailwind CSS can be set up in multiple ways depending on your project's needs, but using **Tailwind CLI** is the simplest method.

Prerequisites

- You need to have **Node.js** and **npm** (or Yarn) installed on your system.
- Basic knowledge of working with the command line.

Step 1: Install Node.js and npm

If you haven't already, download and install Node.js from the official website. Node.js includes npm (Node Package Manager), which you'll use to install Tailwind CSS.

Step 2: Create a New Project Folder

1. Create a folder for your project and navigate into it via the command line.

```
mkdir tailwind-project
cd tailwind-project
```

2. Initialize the project with npm:

```
npm init -y
```

   This will generate a `package.json` file for managing dependencies.

Step 3: Install Tailwind CSS via npm

1. Run the following command to install Tailwind CSS, PostCSS, and Autoprefixer:

```
npm install -D tailwindcss postcss autoprefixer
```

   The `-D` flag installs them as development dependencies.

2. Initialize Tailwind CSS to generate the configuration file:

```
npx tailwindcss init
```

This will create a `tailwind.config.js` file in your project directory.

Step 4: Configure Your Tailwind Setup

1. Open `tailwind.config.js` and update the `content` property to include the paths to all your template files. This tells Tailwind which files to scan for class names.

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.{html,js}"], // Update this path as per your project structure
  theme: {
    extend: {},
  },
  plugins: [],
};
```

In this example, it assumes all HTML and JavaScript files are in a folder named `src`.

Step 5: Create a Tailwind CSS Entry File

1. Inside your project folder, create a folder called `src` (or whatever name you prefer) and a file inside it called `input.css`.

```
mkdir src
touch src/input.css
```

2. Open `src/input.css` and add the following Tailwind directives:

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

```
@tailwind utilities;
```

This imports Tailwind's base styles, component styles, and utility classes.

Step 6: Build the Tailwind CSS File

1. Open the `package.json` file and add a new script in the `"scripts"` section:

```
"scripts": {
  "build": "tailwindcss -i ./src/input.css -o ./dist/output.css --watch"
}
```

This script will compile your Tailwind CSS from `input.css` and output the result into `dist/output.css`.

2. Create a `dist` folder in your project to store the compiled CSS file:

```
mkdir dist
```

3. Run the build script using npm:

```
npm run build
```

This command generates a `dist/output.css` file, which you can use in your HTML.

Step 7: Create an HTML File

1. Create an `index.html` file in your `dist` folder with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tailwind CSS Setup</title>
    <link href="output.css" rel="stylesheet">
</head>
<body class="bg-gray-100 text-center py-10">
    <h1 class="text-3xl font-bold text-blue-600">Hello, Tailwind CSS!</h1>
    <p class="mt-4 text-gray-700">You have successfully set up Tailwind CSS.</p>
    <button class="mt-6 px-4 py-2 bg-blue-500 text-white rounded hover:bg-blue-600">
        Click Me
    </button>
</body>
</html>
```

2. This HTML file includes the compiled CSS (`output.css`) and demonstrates some Tailwind classes like background color, text alignment, padding, and hover effects.

Step 8: Live Preview

1. If you have a live server extension in your code editor (like in VS Code), you can use it to open your `index.html` file for a live preview.

   Alternatively, you can use a simple HTTP server:

   ```
   npx serve dist
   ```

   This command will serve your `dist` folder on a local server (usually at `http://localhost:5000`).

Step 9: Make Changes and See the Results

- You can now modify your `src/input.css` file or your HTML files. Since you are running the `--watch` option in the build script, any changes made to the `input.css` file will automatically update `output.css`.

Tailwind CSS Setup Summary

- **Setup npm** in your project and install Tailwind CSS.
- **Configure Tailwind** in `tailwind.config.js` to watch your HTML and JS files for class names.
- Create a **CSS entry file** with Tailwind's directives.
- **Compile Tailwind** into an output file using `npm run build`.
- Link your **compiled CSS** to your HTML and use Tailwind classes in your HTML.

Next Steps

Once you've set up Tailwind CSS, you can explore its features and classes using the Tailwind CSS Documentation to create responsive and beautiful designs quickly.

1. Tailwind CSS `@apply` Directive

The `@apply` directive in Tailwind CSS allows you to group multiple utility classes into a custom CSS class, making your code more organized and reusable. This is particularly useful when you find yourself applying the same set of utility classes multiple times in different places.

Why Use `@apply`?

- **Reusability**: You can define common styles in a reusable class instead of repeating multiple utility classes across HTML elements.
- **Maintainability**: Easier to maintain and update classes when you centralize them in a single CSS definition.
- **Readability**: It makes your HTML markup cleaner by keeping the class definitions in the CSS file.

Example of `@apply`

Suppose you want to style multiple buttons with the same properties. Instead of repeating all classes for every button, you can use `@apply` to define a reusable class:

```
/* In your input.css or custom CSS file */
.btn {
  @apply bg-blue-500 text-white font-bold py-2 px-4 rounded hover:bg-blue-600;
}
```

Now you can use the `btn` class wherever needed:

```
<!-- In your HTML file -->
<button class="btn">Submit</button>
<button class="btn">Cancel</button>
```

The `@apply` directive in the CSS file will insert the respective utility classes into the generated CSS.

Where to Use `@apply`?

You can use `@apply` in your Tailwind `input.css` or any other custom CSS file you have created. It works with any file that is part of your build process.

Limitations of `@apply`

- The `@apply` directive only works with Tailwind's utility classes. You can't use it with classes that include pseudo-classes (like `hover:`) or variants such as `responsive` within the `@apply`.
- You can't `@apply` classes that are already defined using Tailwind's plugin or component features in the same way.

2. Tailwind CSS Breakpoints

Breakpoints in Tailwind CSS are used to create responsive designs. Tailwind provides default breakpoints that are mobile-first, meaning styles are applied to small screens by default and scale up to larger screens as needed.

Default Breakpoints in Tailwind

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

Each of these breakpoints is associated with a prefix (like `sm:`), which you use in your utility classes to define when the styles should be applied.

How to Use Breakpoints

The syntax for responsive classes is simple: use the breakpoint prefix followed by a colon (`:`) before the utility class.

Example of Responsive Classes

```
<!-- Responsive Container -->
<div class="text-sm sm:text-base md:text-lg lg:text-xl xl:text-2xl">
  This text will change size based on screen width.
</div>
```

Here's what happens:

- On small screens (`sm` or smaller), the text size will be `text-sm`.
- On screens that are 640px or wider (`sm:`), the text size will change to `text-base`.
- On screens that are 768px or wider (`md:`), the text size will change to `text-lg`.
- On screens that are 1024px or wider (`lg:`), the text size will change to `text-xl`.
- On screens that are 1280px or wider (`xl:`), the text size will change to `text-2xl`.

Customizing Breakpoints

You can customize breakpoints by editing your `tailwind.config.js` file. For example:

```
module.exports = {
  theme: {
    extend: {
      screens: {
        'xs': '480px', // Custom screen size for extra small devices
      }
    }
  }
}
```

Now you can use your custom breakpoint with `xs:` like any other default breakpoint.

Summary

- **`@apply` Directive**: Helps group multiple utility classes into a single custom class, improving reusability, readability, and maintainability.
- **Breakpoints**: Used to create responsive designs, allowing you to apply different utility classes based on screen sizes using predefined prefixes (`sm:`, `md:`, `lg:`, etc.).

These features make Tailwind CSS highly flexible and powerful for building responsive and scalable designs.