Tailwind CSS Starter Kit

Created by **JS Mastery**

Visit ***jsmastery.pro*** for more

Introduction

Hey there, fellow developer! Are you tired of the never-ending cycle of tweaking and fine-tuning your CSS styles?

Do you find yourself wrestling with complex class names and struggling to maintain a consistent design across your projects? If you agree, then you're in for a treat!

Welcome to the world of **Tailwind CSS** – the game-changer that's about to revolutionize the way you approach web styling. 🚀

Are you ready to bid farewell to CSS stress and embrace a new era of web design simplicity? The **Tailwind Starter Kit** eBook is your gateway to efficient styling and captivating designs. Join us as we venture into the world of Tailwind CSS, demystifying its power and unleashing its potential.

So, whether you're sipping your morning coffee or burning the midnight oil, dive into this eBook and emerge as a Tailwind CSS virtuoso. The journey starts now – let's create stunning, responsive, & impressive web designs together!

CSS ::before, ::after Selector

The ::before and ::after pseudo-selectors in CSS are used to add content before or after an element without changing the actual HTML structure.

They're great for inserting things like icons, decorative elements, or even text.

These pseudo-elements are like invisible children of the selected element—they only exist in CSS.

How Do They Work?

These pseudo-elements are like invisible children of the selected element—they only exist in CSS.

Both `::before` and `::after` need the content property to work. Even if you just want to insert an empty block or shape, you still need to define content, even if it's just an empty string `("")`.

Here's how they behave:

CSS ::before, ::after Selector

`::before`

Creates content that shows before the main content of an element.

`::after`

Creates content that shows after the main content of an element.

Key Details and Example

Let's break it down with a more detailed example.

Imagine you have a **`<button>`** that you want to style by adding a decorative arrow before the text:

```
<button> Click Me </button>
```

Using **`::before`**, you can insert an arrow before the "Click Me" text like this:

CSS ::before, ::after Selector

```
button::before {
  content: "← "; /* Adds an arrow before the button text */
  color: blue; /* Colors the arrow blue */
  font-size: 25px; /* Makes the arrow bigger */
  margin-right: 8px; /* Adds some space between the arrow
and the text */
}
```

This CSS will add a blue left arrow in front of the "Click Me" text, without touching the HTML.

The content Property

- **Text or Symbols**: You can insert text, symbols, or characters with content. For example, `content: "★";` adds a star symbol.
- **Empty Content**: If you just want to style something (like a background or border) without adding any text, use `content: "";` (empty string).
- **Images**: You can also insert images using `content: url('image.png');`.

CSS ::before, ::after Selector

More Complex Example (with `::after`)

Let's say you want to add an icon after every link to make it clear that it leads to an external site:

Imagine you have a **`<button>`** that you want to style by adding a decorative arrow before the text:

```
<a href="https://example.com">Visit Example</a>
```

With ::after, you can add a small icon after the link:

```
a::after {
  content: "🔗"; /* Adds a link icon after the text */
  margin-right: 5px; /* Adds a little space between the text
and icon */
}
```

Now, the link would look like this: "**Visit Example** 🔗".

CSS ::before, ::after Selector

Block or Inline?

By default, the **`::before`** and **`::after`** pseudo-elements are inline. That means they behave like text—sitting on the same line as the content.

But you can change this using display.

- **Block Elements:** If you want these pseudo-elements to act like block elements (taking up the full width), you can use **`display: block`**.

```
div::after {
  content: "Before the content";
  display: block;
  font-weight: bold;
}
```

This adds the text "Before the content" as a bold, block-level element before a 's content.

Before the content
Hello World

CSS ::before, ::after Selector

Styling Beyond Text

`::before` and `::after` aren't just for adding text. You can style them just like any other element. For example, you can give them:

But you can change this using display.

- **Background:** Add a background color or even a background image.
- **Borders**: Add borders to create decorative lines or boxes.
- **Positioning**: Use `position: absolute` to position the pseudo-elements relative to their parent.

Example: Adding a Decorative Shape

Suppose you want to add a decorative circle before a heading. Here's how you could do it:

```
<h2>My Awesome Heading</h2>
```

CSS ::before, ::after Selector

And the CSS:

```
h2::before {
  content: ""; /* Empty content, we're just adding a shape */
  display: inline-block; /* So it sits inline with the text
  width: 10px;
  height: 10px;
  background-color: red; /* Makes the circle red */
  border-radius: 50%; /* Turns it into a circle */
  margin-right: 10px; /* Adds space between the circle and
the text */
}
```

This will create a small red circle that appears before the heading text, without touching the HTML.

CSS ::before, ::after Selector

Practical Uses

Here are some real-world uses for `::before` and `::after`:

- **Icons:** Add icons before or after text (like arrows, checkmarks, or symbols).
- **Quotes**: Automatically add quote marks around blockquotes without needing to write them in the HTML
- **Clearfix**: A popular technique to clear floats in layouts by adding an empty `::after` element with `clear: both` (it's called a clearfix hack).

- **Buttons or Links**: Add visual cues like arrows, icons, or decorative elements to make buttons and links more engaging
- **Custom Shapes**: Add custom shapes like dots, lines, or borders for styling purposes.

CSS ::before, ::after Selector

Recap

Here are some real-world uses for **`::before`** and **`::after`**:

- **`::before`:** Inserts content before an element's actual content.
- **`::after`:** Inserts content after an element's actual content.
- They only work when you define the **`content`** property, even if it's just an empty string.
- You can style them with any CSS properties (color, size, background, positioning, etc.).
- They're useful for adding decoration, symbols, icons, or additional text without touching the HTML structure.

I hope this gives you a solid understanding of how **`::before`** and **`::after`** work!

Tailwind CSS Before and after

Style the `::before` and `::after` pseudo-elements using the before and after modifiers:

```
<span class="after:content-['*'] after:ml-0.5
after:text-red-500 block...">Email</div>
```

When using these modifiers, Tailwind will automatically add `content: ''` by default so you don't have to specify it unless you want a different value.

Responsive design

Responsive design is a crucial aspect of modern web development that ensures your website looks and functions seamlessly across a variety of screen sizes and devices. With Tailwind CSS, achieving responsive design is made incredibly easy.

Tailwind CSS provides a range of utility classes that can be applied conditionally based on different screen breakpoints. These breakpoints are essentially specific minimum widths that correspond to different device sizes.

Here are the default breakpoints along with their associated prefix and CSS media query:

- `sm` : 640px (min-width: 640px)
- `md` : 768px (min-width: 768px)
- `lg` : 1024px (min-width: 1024px)
- `xl` : 1280px (min-width: 1280px)
- `2xl` : 1536px (min-width: 1536px)

Responsive design

To create responsive elements using Tailwind, you simply prefix the utility class with the breakpoint name followed by a colon. This indicates that the utility class should apply only at or above that specific breakpoint.

It's important to note that Tailwind follows a mobile-first approach. This means that unprefixed utilities, such as those that control basic styles like text alignment and color, apply to all screen sizes by default.

Prefixed utilities, like `md:uppercase`, only take effect at the specified breakpoint and larger.

To illustrate this concept, let's consider creating a responsive card element using Tailwind:

:

Responsive design

```
<div class="mx-auto max-w-md overflow-hidden rounded-xl bg-gray-50 shadow-lg md:max-w-2xl">
    <div class="md:flex">
      <div class="md:shrink-0">
        <img class="h-52 w-full object-cover md:h-full md:w-48" src="/img.jpg" alt="My image alt description" />
      </div>
      <div class="p-8">
        <a href="#" class="mt-1 block text-lg font-bold leading-tight text-gray-800">My awesome card</a>
        <p class="mt-2 text-gray-500">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec accumsan eros elementum massa dignissim.</p>
      </div>
    </div>
  </div>
```

Responsive design

In this example, the `md:flex` class is applied to the outer `<div>`, making it a flex container on medium screens and larger. The `md:shrink-0` class ensures that the image within the card does not shrink on medium and larger screens. Various `md:`-prefixed utilities are used to control the img dimensions & layout on medium & larger screens.

Remember, Tailwind CSS simplifies the process of creating responsive designs by letting you easily manage styles for different breakpoints. This way, your website remains visually appealing and user-friendly across a wide range of devices and screen sizes.

Power of Tailwind CSS

In the ever-evolving landscape of frontend technologies, one name is rising to the forefront: Tailwind CSS. This framework has become synonymous with crafting stunning web interfaces.

Whether you're diving into small demo projects or steering enterprise-level ventures, the potency of Tailwind CSS shines through – and this cheat sheet is your compass on this exhilarating journey.

Tailwind CSS in a Nutshell:

Let's demystify the magic of Tailwind CSS. Imagine styling your web pages without having to wrangle custom CSS code. Tailwind's secret weapon lies in its utility-first approach. It empowers you to effortlessly apply styles using low-level utilities – the fundamental building blocks of any web element. **The best part?**

You don't need to be a CSS guru. Understand the utility classes and watch your designs come to life.

From Vanilla CSS to Tailwind

Visualize a button element – a hallmark of web design. Traditionally, crafting it with vanilla CSS requires a handful of lines. Take a peek at the traditional approach:

```
.button {
  display: inline-block;
  padding: 10px 20px;
  background-color: blue;
  color: white;
  border: none;
  border-radius: 0.25rem;
  cursor: pointer;
  font-size: 16px;
}
```

Now, envision that button, transformed with the Tailwind charm:

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

Tailwind's Class Essentials

1. Categories

Remembering utility classes in Tailwind CSS can be made simpler by grouping them into categories and understanding the naming conventions. Here's a straightforward way to remember and organize them:

Tailwind utility classes can be grouped into a few main categories based on what they control:

Layout

Classes for controlling layout, such as margin, padding, width, height, etc.

Typography

Classes for text-related styles like font size, font weight, text alignment, etc.

Tailwind's Class Essentials

Background & Color

Classes for handling background colors, text colors, borders, etc.

Flexbox & Grid

Classes for building responsive layouts using Flexbox and Grid.

Spacing

Utility classes for managing spacing between elements using margin and padding.

Borders

Classes for styling borders and controlling border radius.

Tailwind's Class Essentials

2. Naming Conventions

Tailwind uses a consistent naming convention for its utility classes. Understanding the abbreviations and structure can help you remember them:

Abbreviations

Each utility class has a short abbreviation indicating what it controls. For example, **`m`** for margin, **`p`** for padding, **`text`** for text-related properties, **`bg`** for background, etc.

Modifiers

Tailwind classes often include modifiers that indicate the level or specific value. For example, **`sm`** for small, **`lg`** for large, **`2xl`** for 2 extra large, **`hover`** for styles on hover, etc.

Property

After the abbreviation and modifier, the property name is added. For example, **`m-2`** for margin-2, **`py-4`** for padding-y-4, **`text-center`** for center-aligned text, etc.

Tailwind's Class Essentials

3. Examples

Here are a few examples to illustrate this approach:

`m-4`

Margin of 1rem (default spacing unit) on all sides.

`py-8`

Vertical padding of 2rem on top and bottom.

`text-lg`

Large font size for text.

`bg-blue-500`

Background color of a shade of blue.

`flex flex-col`

Apply Flexbox layout with column direction.

`border border-gray-300`

Gray border with a default width.

Tailwind's Class Essentials

By understanding the categories, abbreviations, modifiers, and property names, you can quickly recall and apply Tailwind utility classes without needing to memorize every single class.

It's all about breaking down the naming logic and knowing where to find the classes you need for a specific styling task.

Tailwind Play

One of the easiest ways to get started with Tailwind is by using its interactive web browser mode on Tailwind Play.

This fantastic feature enables you to make updates to an existing HTML page and instantly visualize the changes. It's a perfect starting point, along with the comprehensive Tailwind documentation, to explore various features and get familiar with the Tailwind approach.

With this interactive playground, you can quickly experiment and see the power of Tailwind CSS in action!

Dynamic States

You've Embraced Tailwind CSS – Now Embrace Its Dynamic States!

Tailwind CSS is your go-to toolkit for crafting stylish and responsive interfaces with ease. But in today's world, where interactivity and adaptability reign supreme, static designs just won't cut it. Modern applications call for dynamic responses to user interactions and varying screen sizes.

Forget the headaches of traditional CSS frameworks! Tailwind CSS empowers you to effortlessly implement dynamic element states by harnessing the magic of CSS pseudo-selectors as prefixes to its intuitive classes.

Need a captivating **hover effect**? Transform an element's background on hover with a simple "**hover**" prefix:

```
<div class="bg-gray-500 hover:bg-blue-600 ...">Hover
Me</div>
```

Dynamic States

But wait, there's more! Tailwind CSS extends its prowess beyond just "hover".

It embraces an array of pseudo-selectors that grant you incredible versatility. For those complex scenarios, meet the mighty "group" class. It empowers you to apply styles to child elements based on the parent's state:

```
<div class="bg-blue-500 group ...">
    <p class="text-blue-300 group-hover:text-white">
Click for more</p>
</div>
```

Elevate your design game with Tailwind CSS – Where dynamic states are a breeze!

Dark Mode Enhancement

Tailwind CSS not only excels at responsive design but also seamlessly integrates dark mode styling into your web projects. With the dark mode variant, you can effortlessly adapt your website's appearance when users switch to dark mode.

Consider the same card example, now enhanced with dark mode styling:

Dark Mode Enhancement

Just like everything else, setting up and styling for dark and light themes is easier than before.

To use dark mode, we need to make a small change. We have to modify the configuration to include the dark class.

```
/** @type {import('tailwindcss').Config} **/
module.exports = {
  darkMode: 'class',
  theme: {
    extend: {
      // ...
    },
  },
  plugins: [],
}
```

If we don't specify it, the theme styles will automatically adapt based on the user's Operating System preference – quite interesting!

We only need to mention this field if we want to offer a theme toggle on the website. It's a choice we make!

Dark Mode Enhancement

```
<div class="mx-auto max-w-md overflow-hidden rounded-xl bg-gray-50 shadow-lg dark:bg-slate-800 md:max-w-2xl">
    <div class="md:flex">
      <div class="md:shrink-0">
        <img class="h-52 w-full object-cover md:h-full md:w-48" src="/img.jpg" alt="My image alt description" />
      </div>
      <div class="p-8">
        <a href="#" class="mt-1 block text-lg font-bold leading-tight text-gray-800 dark:text-white">My awesome card</a>
        <p class="mt-2 text-gray-500 dark:text-gray-300">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec accumsan eros elementum massa dignissim.</p>
      </div>
    </div>
</div>
```

In this updated example, the dark mode variant comes into play. Let's break it down:

:

Dark Mode Enhancement

- **`dark:bg-slate-800`**: This utility sets the background color of the card to a darker shade (bg-slate-800) when dark mode is enabled. This creates a visually distinct experience for users who prefer dark themes.

- **`dark:text-white`**: Here, the link text inside the card is set to white (text-white) in dark mode. This ensures good readability and maintains a consistent design aesthetic.

- **`dark:text-gray-300`**: The paragraph text is given a gray color (text-gray-300) to provide a suitable contrast while maintaining the dark mode look.

Tailwind makes dark mode implementation hassle-free by automatically detecting the user's preferred color scheme through the prefers-color-scheme CSS media feature.

By incorporating these dark mode utilities, you enhance the UX and demonstrate your attention to detail by providing a design that adapts to different viewing preferences.

:

Functions & Directives

Alright, gather 'round as we dive into the mesmerizing world of Tailwind CSS – where functions and directives work their enchantment on your styles.

Get ready for some behind-the-scenes wizardry that'll have your CSS dancing to your tune!

Directives: The Ringleaders of Style

Let's talk directives – these are Tailwind's special commands that make your CSS do a little jig.

Tailwind Directives: A Symphony of Style:

`@tailwind:`

This big boss directs the orchestra. It brings in Tailwind's base, components, utilities, and variants styles into your CSS. It's like your backstage pass to the styling show.

:

Functions & Directives

```
 /* Injects base styles and plugin base styles */
@tailwind base;
 /* Injects component classes and plugin component styles */
@tailwind components;
/* Injects utility classes and plugin utility styles */
@tailwind utilities;
/* Controls where variants are injected */
@tailwind variants;
```

`@layer:`

Meet the director of categories – base, components, and utilities. You get to tell your styles which bucket to hop into.

```
@layer base {
  /* Your base styles here */
}

@layer components {
  /* Your component styles here */
}

@layer utilities {
  /* Your utility styles here */
}
```

Functions & Directives

@apply:

This one's a ninja. It lets you sneak in existing utility classes into your custom CSS. Think of it like mixing and matching style ingredients.

```
.custom-button {
  @apply bg-blue-500 text-white font-bold py-2 px-4 rounded;
}
```

@config:

Need a CSS file to follow different rules? Wave the @config wand and let Tailwind know which config file to dance with.

```
@config "./tailwind.admin.config.js";

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

:

Functions & Directives

Functions: CSS Sorcery Unleashed

Now, let's talk functions. These are like spells that give you dynamic powers over your styles.

`theme()`

This wizard lets you access Tailwind config values using dot notation. It's like whispering secrets to your styles.

```
.content-area {
  height: calc(100vh - theme(spacing.12));
}
```

Got a value with a dot? No problem – use square brackets.

```
.content-area {
  height: calc(100vh - theme(spacing[2.5]));
}
```

:

Functions & Directives

screen()

Now, meet the maestro of media queries. It crafts these responsive symphonies that perfectly match your breakpoints.

```
@media screen(sm) {
  /* Media query for the 'sm' breakpoint */
}
```

With screen(), you compose media queries that rock to the rhythm of your breakpoints.

So, there you have it – functions and directives, the secret ingredients that make Tailwind CSS so much more than just styling.

Flexibility and Extendibility

In the **tailwind.config.js file**, you'll discover the cool **theme** and **plugin** fields. These are your tickets to unlocking Tailwind CSS's awesomeness, letting you tweak it just the way you want.

Theme

The **theme** part lets you supercharge Tailwind CSS by adding your own stuff, like new colors or spacing. When you do this, Tailwind CSS automatically creates classes that work seamlessly with the ones it already has.

You can even change the values it comes with to make Tailwind totally your own.

Want a taste of how this works? Check out this snippet from your tailwind.config.js file. Here, we're switching up the default sans fontFamily and throwing in a new color we're calling nightown.

:

Flexibility and Extendibility

```
module.exports = {
  theme: {
    fontFamily: {
      sans: ['YourCustomFont', 'system-ui', 'sans-serif'],
    },
    extend: {
      colors: {
        nightown: '#123456',
      },
    },
  },
};
```

And Now, Plugins!

Plugins are like the cherry on top that makes Tailwind even more awesome. Let's highlight two standout plugins: typography and forms.

Typography gives you classes to make your content look all fancy with Markdown text—super useful when you're working with HTML generated by CMS.

:

Flexibility and Extendibility

Forms are where it's at for styling your form elements like **input**, **textarea**, and **select**. They come pre-styled so your forms look slick right out of the box.

If you want to take control, you can slap on the **form-input** classes to choose where these styles do their thing.

With Tailwind CSS, you're in for a ride of endless customization and style expansion. It's all about making your styles your own, and Tailwind's got your back.

:

Accessibility in Tailwind

Tailwind has your back when it comes to accessibility, especially for screen readers. It provides nifty utilities like **`sr-only`** and **`not-sr-only`** classes to ensure a seamless experience for all users.

The Invisible Hero: `sr-only` Class

Imagine an **`<a>`** tag with an SVG icon and a hidden text for screen readers.

```
<a href="#">
  <svg><!-- ... --></svg>
  <span class="sr-only">User Profile</span>
</a>
```

The **`sr-only`** class does the magic – it makes the "User Profile" text invisible to the naked eye but still there for those who rely on screen readers.

:

Accessibility in Tailwind

Hiding on Demand: `not-sr-only` Class

Now, let's say you want to show the text for larger screens but keep it hidden for smaller ones. The **`sm:not-sr-only`** class combo is your ticket to this flex.

```
<a href="#">
  <svg><!-- ... --></svg>
  <span class="sr-only sm:not-sr-only">User Profile</span>
</a>
```

With this, the text remains hidden for small screens, but magically appears on larger ones.

Creating Bridges to Inclusion:

Tailwind's accessibility utilities are like bridges that connect all users to your content.

With **`sr-only`** and **`not-sr-only`**, you're weaving a tapestry of inclusivity where everyone gets a front-row seat, no matter how they experience your site.

Animations and Interactivity

Let's venture into the world of animations and interactivity, where Tailwind CSS truly shines. Get ready to add dynamic flair to your designs!

Transitions

Start with smooth transitions using the **`transition-{properties}`** utilities. These help you specify which properties should transition when they change. For eg:

```
<button class="transition ease-in-out delay-150 bg-blue-500
hover:-translate-y-1 hover:scale-110 hover:bg-indigo-500
duration-300 ...">
  Download
</button>
```

Animations

Tailwind offers four standard animations to give life to your elements:

:

Animations and Interactivity

Spin Animation

Create a spinning animation, like a loading indicator, with the animate-spin utility. Example:

```
<div class="animate-spin h-6 w-6 ..."></div>
```

Ping Animation

Make an element scale and fade like a radar ping animation using animate-ping. Example:

```
<div class="animate-spin h-6 w-6 ..."></div>
```

Pulse Animation

Add a gentle fade-in and fade-out animation to an element with animate-pulse. Example:

```
<div class="animate-pulse h-8 w-8 rounded-full ..."></div>
```

Animations and Interactivity

Bounce Animation

Use animate-bounce to make an element bounce up and down. Example:

```
<div class="animate-bounce h-10 w-10 rounded ..."></div>
```

Interactivity

Tailwind makes your elements interactive and engaging:

Cursor Styles

Control cursor appearance on hover:

```
<button class="cursor-pointer ...">Click me</button>
<button class="cursor-wait ...">Please wait</button>
<button class="cursor-not-allowed ...">No access</button>
```

:

Animations and Interactivity

Pointer Events

Enable or disable pointer interactions:

```
<div class="pointer-events-auto ...">Clickable</div>
<div class="pointer-events-none ...">Not clickable</div>
```

With these animations and interactive elements, Tailwind CSS lets you create engaging and user-friendly designs effortlessly. It's all about adding that extra layer of magic to your web projects!

:

Tips & Tricks

Now that you have understood, what tailwindcss is and how it works, let's explore a few tips & tricks.

We'll first explore the tricks or special utilities we can use to make our development more efficient:

Accent

Changes the default browser color for elements like checkboxes and radio groups

```
<div class="my-4 flex flex-col">
  <label> <input type="checkbox" checked /> Browser default
</label>
  <label> <input type="checkbox" class="accent-pink-500"
checked /> Customized </label>
</div>
```

- [x] Browser default
- [x] Customized

Tips & Tricks

Fluid Texts

Usually, when building a responsive website, we have to take care of the text size on different devices. Something like this:

```
<p class="sm:text-7xl text-5xl">Something Nice</p>
```

Using the custom style approach we discussed and the **`min`** CSS function, we can do something like this:

```
<p class="text-[min(10vw,70px)]">Something Fluid</p>
```

The result, the second approach is much better and more responsive than the other. Instead of relying on the media screen, we kind of calculate the value depending on the screen size automatically.

Something Nice

Something Fluid

Something Nice

Something Fluid

Tips & Tricks

Less JavaScript

Well, yeah. If you take the time to learn it well, you can perform certain tasks that are typically done using JavaScript with tailwindcss alone, without requiring JavaScript. It might sound incredible, right?

Think about how often we've made an accordion or relied on libraries or states to manage it.

Let me demonstrate an example of how we can achieve this without using JavaScript, just with pure tailwindcss.

```
<div class="max-w-lg mx-auto p-8">
  <details class="open:bg-white dark:open:bg-slate-900 open:ring-1
open:ring-black/5 dark:open:ring-white/10 open:shadow-lg p-6
rounded-lg" open>
    <summary class="text-sm leading-6 text-slate-900 dark:text-white
font-semibold select-none">
      Why do they call it Ovaltine?
    </summary>
    <div class="mt-3 text-sm leading-6 text-slate-600 dark:text-
slate-400">
      <p>The mug is round. The jar is round. They should call it
Roundtine.</p>
    </div>
  </details>
</div>
```

Tips & Tricks

▸ **Why do they call it Ovaltine?**

▾ **Why do they call it Ovaltine?**

The mug is round. The jar is round. They should call it Roundtine.

Isn't that crazy???

We used the open: selector along with proper HTML tags, i.e., details (without which we won't get that toggle).

File

If you worked with file inputs, you know the pain of styling the default layout.

Thankfully, Tailwindcss provides a better solution

Use the `file:` prefix and use any utility to customize the input however you want

Tips & Tricks

```
<label class="my-4 block">
  <input type="file" class="block w-full text-sm text-slate-500
file:mr-4 file:rounded-full file:border-0 file:bg-violet-50
file:px-4 file:py-2 file:text-sm file:font-semibold file:text-
violet-700 hover:file:bg-violet-100" />
</label>
```

Highlight

Do you want to override the default blue or other highlight that appears when a user selects a text on your website? Tailwindcss's selection is a way to go

```
<div class="selection:bg-green-400 selection:text-white">
   <p>Subscribe to JavaScript Mastery YT channel</p>
</div>
```

Subscribe to JavaScript Mastery YT channel

Be creative, do whatever you want. You have to complete control by using just a few lines of code

Tips & Tricks

Caret

Not a fan of white or black caret? Use tailwindcss then

```
<textarea class="w-full border border-zinc-200 caret-pink-500 p-2" placeholder="type something.."></textarea>
```

Try typing it into textarea, and you'll notice the caret color to be pink 😃

Type something..

Many More

These are just a handful of examples. Tailwindcss offers many unique tools such as those for different states like before: & active:, styles that work only on certain screen sizes like landscape or portrait, styles for ARIA and screen readers, gradients animations, and it even lets you apply distinct styles when printing.

Tailwind Component Libraries

tailwind UI

Visit ↗

A premium collection of meticulously crafted user interface components and templates built using the Tailwind CSS framework.

headless UI

Visit ↗

A set of unstyled, fully accessible UI components that can be used as a foundation for creating custom designs. Built by the creators of Tailwind CSS.

Shadcn

Visit ↗

Shadcn combines the power of Tailwind with artistic designs. Beautifully designed components built with Radix UI and Tailwind CSS.

Tailwind Component Libraries

Tailkit

Visit ↗

Tail-kit provides a comprehensive set of carefully crafted, easy to customize, fully responsive UI components, Templates & Tools for your Tailwind CSS.

Radix

Visit ↗

Radix UI is a cutting-edge toolkit designed to create advanced UI components with an emphasis on accessibility and user experience.

preline

Visit ↗

Preline UI is an open-source set of prebuilt UI components based on the utility-first Tailwind CSS framework. With a focus on high-quality design.

Tailwind Component Libraries

daisyUI

Visit ↗

The most popular component library for Tailwind CSS. daisyUI adds component class names to Tailwind CSS so you can make beautiful websites faster than ever.

Hyper UI

Visit ↗

Explore an extensive array of ready-to-use components, organized into three distinct categories: marketing, e-commerce, and applications.

Sailboat UI

Visit ↗

Contemporary Tailwind CSS component library with 150+ rich components for app & product development. Each offers multiple variations to suit your requirement

Tailwind Component Libraries

Tailwind Elements

Visit ↗

With more than 500 tailwind components. These components are based on the Bootstrap framework, but they have a better design & a lot more functional.

Tailwind Templates

Visit ↗

This is a library that contains a wide variety of free and paid templates. They have a collection of around 25 templates, mostly geared toward startup & saas.

Tailblocks

Visit ↗

A collection of predefined Tailwind CSS components designed to facilitate rapid website prototyping and development.

Tailwind Component Libraries

MAMBA UI

Visit ↗

It offers a wide selection of 150+ Tailwind components and templates with versatile styles. It's adaptable for various frameworks like Angular, Vue, React, Svelte, etc.

Sira

Visit ↗

Sira a design system with reusable components. Compatible with Vue, React, and more. Offers themes, dark mode, & predefined Tailwind styles.

TailGrids

Visit ↗

Containing over 500 complimentary and premium components, blocks, sections, and templates, this kit offers an extensive selection.

The End

Congratulations on reaching the end of our guide! But hey, learning doesn't have to stop here.

If you're eager to dive deep into something this specific and build substantial projects, our **special course on Next.js** has got you covered.

The Ultimate Next 14 Course

If you're craving a more personalized learning experience with the guidance of expert mentors, we have something for you — [Dev Career Accelerator](#).

Dev Career Accelerator

If this sounds like something you need, then don't stop yourself from leveling up your skills from junior to senior.

Keep the learning momentum going. Cheers! 🚀