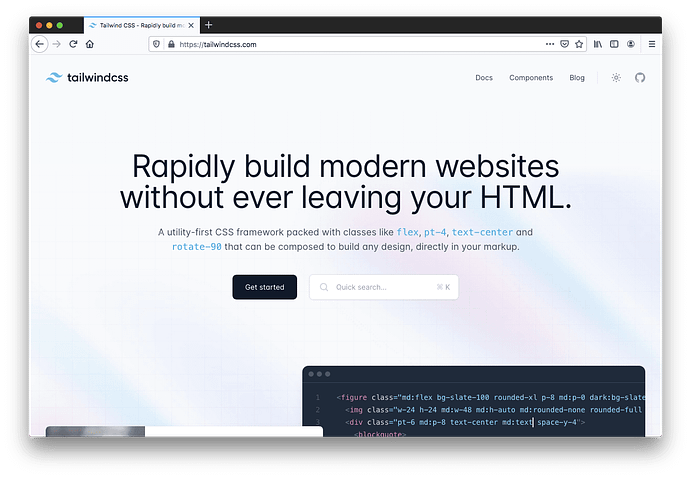
**Tailwind CSS Tutorial for Beginners: A Guide to Get Started in 2023**

**📚 Table of contents:**

1. Background: Atomic CSS [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#background-atomic-css)
2. Why Atomic CSS? [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#why-atomic-css)
3. What is Tailwind CSS? [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#what-is-tailwind-css)
4. Getting started in Tailwind developer mode [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#getting-started-in-tailwind-developer-mode)
5. Using Tailwind’s utility classes [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#using-tailwinds-utility-classes)
6. Learning Tailwind’s “language” [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#learning-tailwinds-language)
7. Grouping repeated styles with Tailwind [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#grouping-repeated-styles-with-tailwind)
8. Installing Tailwind’s full toolset [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#installing-tailwinds-full-toolset)
9. Configuring Tailwind projects [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#configuring-tailwind-projects)
10. Generating a Tailwind build [#](https://www.codeinwp.com/blog/tailwind-css-tutorial/#generating-a-tailwind-build)



**Background: Atomic CSS**

In order to understand Tailwind a little better, let’s back up a bit to ensure this Tailwind CSS tutorial makes sense to you.

Tailwind’s concepts are rooted in what’s commonly referred to as *Atomic CSS*, sometimes referred to as *CSS utility classes*. This is a CSS concept where a single HTML class applies a single CSS property/value pair to an element. Here’s a few examples:

**.font-weight-bold {**

**font-weight: bold;**

**}**

**.color-hotpink {**

**color: hotpink;**

**}**

Then in your HTML you’d use the classes:

**<p class="font-weight-bold color-hotpink">Some example text.</p>**

**Code language: HTML, XML (xml)**

**Why Atomic CSS?**

When you have access to every possible property/value pair by means of single-purpose utility classes, like those shown in the previous example, you have a few benefits:

* You don’t have to worry about CSS specificity. Every “utility” is a class name, so they’re all at the same level of specificity.
* No need to write any CSS at all; you only build elements with class names in the HTML.
* You can build just about anything with nothing but HTML.

On the surface, it can seem as though atomic CSS and libraries like [Tailwind CSS](https://tailwindcss.com/) make your HTML large and somewhat unwieldy. But that’s a minor hindrance in comparison to the benefits I mentioned above.

That should be enough of an introduction to atomic CSS.

**What is Tailwind CSS?**

Tailwind is a library of atomic CSS rules (i.e., single-purpose utility classes) that helps you build HTML pages without touching your CSS. But Tailwind isn’t just the CSS. In addition to the framework itself, Tailwind includes a [CLI](https://www.codeinwp.com/blog/wp-cli/) and various configuration and theming options. These allow you to automate and scale your builds, thus making the most of Tailwind’s features without sacrificing performance and maintainability.

Before I get to an explanation of the CLI and build options in this Tailwind CSS tutorial for beginners, let’s first consider features of the library itself (i.e. the atomic classes). This will demonstrate how easy it is to build stuff with Tailwind.

**Getting started in Tailwind developer mode**

Since Tailwind is designed to allow you to build pages without touching your CSS, this naturally makes the full Tailwind library a large file that you would *never* want to include in a real website build.

For the purposes of this Tailwind CSS tutorial, in order to learn and experiment with the library, you can use the following code in the <head> of your HTML to include all of Tailwind’s utility classes:

**<head>**

**<meta charset="UTF-8">**

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

**<script src="https://cdn.tailwindcss.com"></script>**

**</head>**

**Code language: HTML, XML (xml)**

The above code includes a script referred to as the [Tailwind Play CDN](https://tailwindcss.com/docs/installation/play-cdn). *This script is only to be used for learning and experimenting, never for a live site*. I’ll discuss later how to properly include Tailwind in your projects via the build process. For now, this will do for trying Tailwind out and starting to add classes to style HTML elements.

**Using Tailwind’s utility classes**

Once you have Tailwind’s library of utilities accessible to a particular page, you can start adding classes to whatever element you want to style.

Here’s a CodePen example with a few page elements, including a button, styled with some Tailwind classes:

The HTML looks like this (text removed for brevity):

**<main class="my-0 mx-auto max-w-3xl text-center">**

**<h2 class="p-6 text-4xl">...</h2>**

**<p class="px-10 pb-10 text-left">...</p>**

**<button class="bg-sky-600 hover:bg-sky-700 px-5 py-3 text-white rounded-lg">...</button>**

**</main>**

**Code language: HTML, XML (xml)**

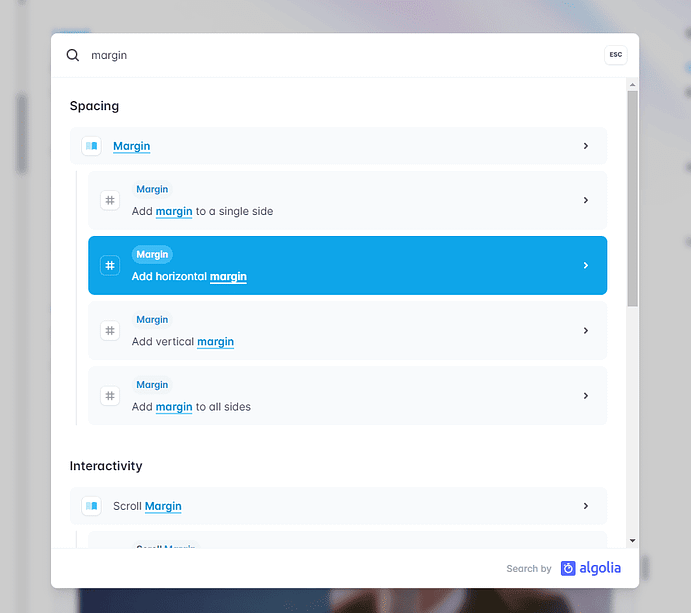
As you can see, Tailwind’s utility classes more or less have their own language, based on each of the property/value pairs they represent in natural CSS.

This can be a little intimidating at first, and it’s certainly not for everyone. But many developers have found that this, along with Tailwind’s tooling, is a scalable and efficient way to build large projects that are easier to maintain.

**Learning Tailwind’s “language”**

There’s no way that a Tailwind CSS tutorial for beginners like this could possibly cover all the different nuances of how the utility classes are named. That’s something you’ll have to learn over time, but I’ll do my best to give you some resources to help out.

The Tailwind docs provide lots of info on the class names. You can start in [the Layout section](https://tailwindcss.com/docs/aspect-ratio) and work your way through. Any time you want to add a specific CSS property to an element and aren’t sure of the value, you can use the interactive on-page search feature. It’s available at the top of the left navigation or by using CTRL-K (CMD-K) on your keyboard.



Type the keyword you want (e.g., the CSS property you want to use) and the search modal will show you relevant results from the docs.

Some examples of common patterns found in the Tailwind utility class syntax include:

* Abbreviations, like w, h, sm, md, and lg, for sizing
* Near-literal CSS equivalents, like float-left, clear-both, overflow-auto, and so forth
* Vertical and horizontal indicators using x and y
* Min/max sizing indicators, like min-w and max-h
* Font stack abbreviations, like font-sans and font-mono

That’s just a small sampling. Tailwind covers everything in CSS including utilities for all features related to [Flexbox](https://www.codeinwp.com/blog/css-flexbox-tutorial/) and [CSS Grid](https://www.codeinwp.com/blog/css-grid-tutorial-layout/). As mentioned, there’s no way to cover it all here but this should be enough to give you an idea of the kinds of classes you’ll be adding to your elements. If you’ve used a more traditional CSS framework, like [Bootstrap](https://www.codeinwp.com/blog/best-free-bootstrap-templates-for-reactjs/), then you’ve already done this sort of thing on a less-granular level. Tailwind takes this to a more atomic level, for more fine-grained control over your styles, almost as if you’re writing CSS directly.

Of course, you’ll never remember all the class names, so here are a few tools that might help:

* [Tailwind Cheat Sheet](https://tailwindcomponents.com/cheatsheet/) on the Tailwind Components website.
* [Tailwind CSS Cheatsheet](https://github.com/gojutin/tailwind-cheatsheet), a printable cheat sheet of Tailwind’s classes
* [Tailwindcss-cheatsheet](https://umeshmk.github.io/Tailwindcss-cheatsheet/) by Umesh Kadam

**Grouping repeated styles with Tailwind**

Earlier, I showed you an example of HTML with Tailwind utility classes littered throughout. You may have been disturbed by how inefficient it would be to have to style every single element that way – even if the same styles are repeated!

For example, what if I had five buttons on the same page that all looked like this:

**<button class="bg-sky-600 hover:bg-sky-700 px-5 py-3 text-white rounded-lg">...</button>**

**Code language: HTML, XML (xml)**

If the buttons appeared one after the other, then I could take advantage of multi-cursor editing in my text editor if I needed to make any changes. I could alternatively use a templating language (either on the front-end or back-end) to dynamically include the repeated elements, which means I would only have to change them in one spot. But what if the elements aren’t that close together or are on separate pages and the above solutions don’t help?

For repeating sections across projects, Tailwind’s docs [strongly recommend](https://tailwindcss.com/docs/reusing-styles#extracting-components-and-partials) building your pages by taking advantage of components in libraries like React or template partials in libraries like Twig. This means your styles will be 100% built with utility classes alone, which is ideal.

That being said, Tailwind has [directives](https://tailwindcss.com/docs/functions-and-directives) called @tailwind, @layer, and @apply that allow you to create smaller, reusable collections of utility classes. These would be components (often a single element) that are too small for a full-blown library-level abstraction, but too unwieldy to have to repeat elsewhere (similar to the button example above).

Using the aforementioned directives, I can convert the button example to the following simple HTML:

**<button class="btn-main">...</button>**

**Code language: HTML, XML (xml)**

This single HTML class represents all the utility classes I had previously included on my button. This is accomplished using the following in my CSS:

**@tailwind base;**

**@tailwind components;**

**@tailwind utilities;**

**@layer components {**

**.btn-main {**

**@apply bg-sky-600 hover:bg-sky-700 px-5 py-3 text-white rounded-lg;**

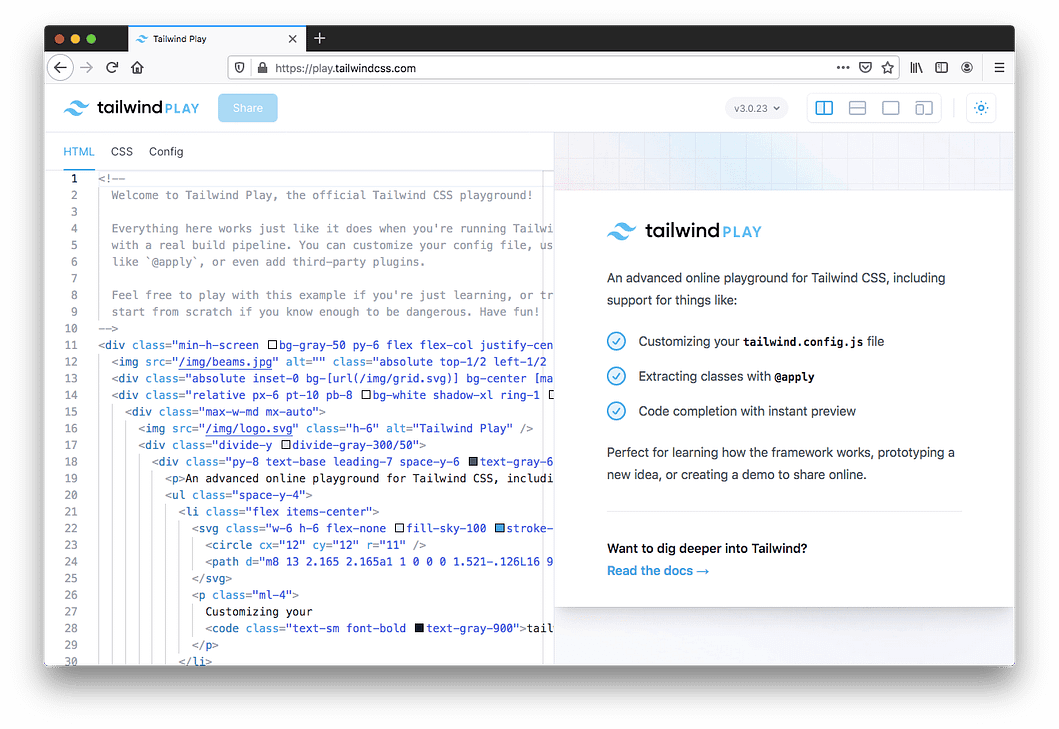
**}**

**}**

Notice how I’ve moved all the utility classes to the line that starts with @apply. The code above does the following:

* Inserts Tailwind’s base, components, and utilities styles into the CSS
* Includes the btn-main class in the components “bucket” using @layer
* Assigns the selected utility classes to the btn-main class name

With that, I can use .btn-main anywhere I want and all buttons will inherit the same styles. You can see this in action on [this demo](https://play.tailwindcss.com/afODYquLr4) which I created using Tailwind’s official playground. Click the **CSS** tab to view the code that encapsulates all the button classes into .btn-main.



**Installing Tailwind’s full toolset**

As mentioned, the examples I’ve shown you up to this point are not ideal. The CodePen, for instance, uses the Tailwind Play CDN, which means all of Tailwind’s classes are included in the page, even though I’m only using a few.

To use Tailwind properly, let’s install it and configure it to build only the CSS that I need. In my project folder I’m going to run the following two commands in my terminal:

**npm install -D tailwindcss**

This installs Tailwind along with all its dependencies in my project folder. Next, I want to initiate Tailwind to create my config file:

**npx tailwindcss init**

This generates a file called tailwind.config.js in the root of my project folder with the following content inside:

**module.exports = {**

**content: [],**

**theme: {**

**extend: {},**

**},**

**plugins: [],**

**}**

**Code language: JavaScript (javascript)**

Adding anything to this file is optional. If you leave any parts blank, Tailwind will use its default configuration options, which you can [view here](https://github.com/tailwindlabs/tailwindcss/blob/master/stubs/defaultConfig.stub.js). The Tailwind docs recommend filling in only the options you want changed, rather than including the full default config file (which you can do by using npx tailwindcss init --full when initiating the project).

[Go to top](https://www.codeinwp.com/blog/tailwind-css-tutorial/)

**Configuring Tailwind projects**

Going into advanced customizations is beyond the scope of this Tailwind CSS tutorial for beginners, but here’s a basic outline of some of the things you can do in the config file:

* content – Define paths to your HTML templates, JavaScript components, and other files that use Tailwind class names.
* theme – Define default theming options (colors, fonts, etc.) related to the design of your pages and components.
* plugins – Include plugins that extend Tailwind.
* presets – Define your own base configuration, instead of using Tailwind’s default.
* prefix – Define a custom prefix that will appear in front of every Tailwind utility class. This can be useful to overcome class naming conflicts.
* important – Forces all of Tailwind’s utility classes to be marked with !important, which might help if you’re adding Tailwind to a legacy project that uses a lot of high-specificity selectors.

For full info on the available configuration options, see [the Tailwind docs](https://tailwindcss.com/docs/configuration).

For demonstration purposes, let’s assume a simple configuration file that defines the location of my content:

**module.exports = {**

**content: [**

**'./pages/\*\*/\*.{html,js}',**

**'./components/\*\*/\*.{html,js}'**

**],**

**theme: {**

**extend: {},**

**},**

**plugins: [],**

**}**

**Code language: JavaScript (javascript)**

This tells Tailwind that I’m including Tailwind classes inside .html and .js files within two folders: pages and components. I’m including JavaScript files because Tailwind recognizes classes added and removed dynamically via JavaScript, some of which will be present only in the .js files and not in the HTML.

In some cases, I might use a src folder, common in many projects, with a public HTML entry point. That would look like this:

**module.exports = {**

**content: [**

**'./public/index.html',**

**'./src/\*\*/\*.{html,js}',**

**],**

**theme: {**

**extend: {},**

**},**

**plugins: [],**

**}**

**Code language: JavaScript (javascript)**

It all depends on how you’ve structured your project and the associated content files. You also might be using a processor, like PostCSS, which you can [read about here](https://tailwindcss.com/docs/using-with-preprocessors).

As a warning: Make sure you aren’t overly general when identifying content pages in your configuration. For example, you don’t want to use wildcards everywhere, thus making Tailwind look inside node\_modules or other folders it shouldn’t be traversing.

**Generating a Tailwind build**

Once you’ve added all your classes to your HTML and JavaScript pages and you’ve identified the location of the pages in Tailwind’s configuration options, you want to build your project to only include the CSS that you’ve specified.

For my example, I’m going to run the following command in my project’s root folder:

npx tailwindcss -o ./css/styles.min.css --minify

This uses the -o command to tell Tailwind to output my CSS in a minified format in the specified folder. This ensures that no unused CSS will exist in my project. If I’ve only used 30 Tailwind classes in my HTML, then only those utilities will be generated in my CSS. From there, I just have to make sure that any HTML pages that use Tailwind classes include a reference to my stylesheet.

Of course, this is just one example of a build and folder structure. Depending on whether you’re using a [library](https://www.codeinwp.com/blog/best-javascript-libraries/) like React or [Vue](https://www.codeinwp.com/blog/vue-ui-component-libraries/), or if you’re scaffolding a particular custom setup each time, you’ll make the necessary adjustments in your configuration file and any associated npm commands during your build step.

**Tailwind CSS tutorial conclusion**

That should be enough to get you up and running with Tailwind before getting into more complex builds. You have the option to use the developer-only Play CDN to experiment and learn the Tailwind utility class language. When it’s time for your build step, you can configure Tailwind to read your HTML and JavaScript files to build only the CSS you’re using.

Once you have these basic concepts in place, you can start incorporating different Tailwind tools, taking advantage of Tailwind’s [responsive design](https://www.codeinwp.com/blog/responsive-images/) features, using various [component libraries](https://www.codeinwp.com/blog/tailwind-css-templates/), plugins, and more. Is Tailwind for everyone? No, and it’s probably somewhat of an overkill for smaller projects. I wouldn’t recommend using Tailwind unless you first learn CSS. But what I’ve presented here should be enough to get you up and running with a Tailwind project today.