

@oluwakemi oluwadahunsi

# 12 Unique Features of



# Tailwind CSS

# Introduction

Say goodbye to clunky CSS files and endless debugging!

Switching to **Tailwind CSS** has revolutionized the way I style web applications. Its utility-first approach, combined with speed, flexibility, and responsiveness, makes it a must-have tool for modern frontend developers.

**What is Tailwind CSS?** Tailwind CSS is a **utility-first** CSS framework that provides low-level classes for designing user interfaces.

Instead of using **predefined components**, it allows you to build custom designs quickly by combining these utility classes directly in your HTML, offering great flexibility and speed in web development.

Let's talk about features that makes Tailwind CSS easier to use; about 12 of them:

# 1. Utility-first Classes


These are the building blocks that promise ultimate control over design while maintaining a tidy stylesheet.

Tailwind CSS provides **low-level utility classes** that let you build any design directly in your **HTML**. No need to write **custom CSS** for every component!

For example: Styling a button using conventional CSS

```
1 .button {  
2   background-color: blue;  
3   color: white;  
4   font-weight: bold;  
5   padding: 0.5rem 1rem;  
6   border-radius: 0.25rem  
7 }
```

Same style with Tailwind CSS:



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

From the examples above, the tailwind utility classes not only makes it easier to declare styles for elements but also makes coding faster.

what this code means:

- **bg-blue-500**: Sets a blue background.
- **text-white**: Makes the text white.
- **py-2**: Adds padding top and bottom.
- **px-4**: Adds padding left and right.
- **rounded**: Adds border radius for rounded corners.

This direct manipulation of styles, without flipping back and forth between HTML and CSS files, significantly refines the development workflow.

## 2. Responsive Design Made Easy

**Responsive design** ensures that web applications adapt seamlessly to various screen sizes and devices, providing an optimal user experience across desktops, tablets, and mobile phones.

**Tailwind CSS** simplifies responsive design with a built-in, intuitive approach using **breakpoints** and **utility classes**.

Tailwind uses **breakpoints** that correspond to common screen sizes. Each breakpoint has a **prefix** that you can attach to utility classes to make them apply at specific **screen widths**.

Breakpoints in Tailwind CSS:

Prefix	Minimum width	Device Type
sm	640px	Small screens (mobile)
md	768px	Medium screens (tablet)
lg	1024px	Large screens (laptops)
xl	1280px	Extra-large screens (desktop)
2xl	1536px	Ultra-large screens (widescreen monitors)

Examples of Responsive Design in Tailwind:

```

1 <p class="text-sm sm:text-base md:text-lg lg:text-xl">
2   Responsive Font Sizes
3 </p>

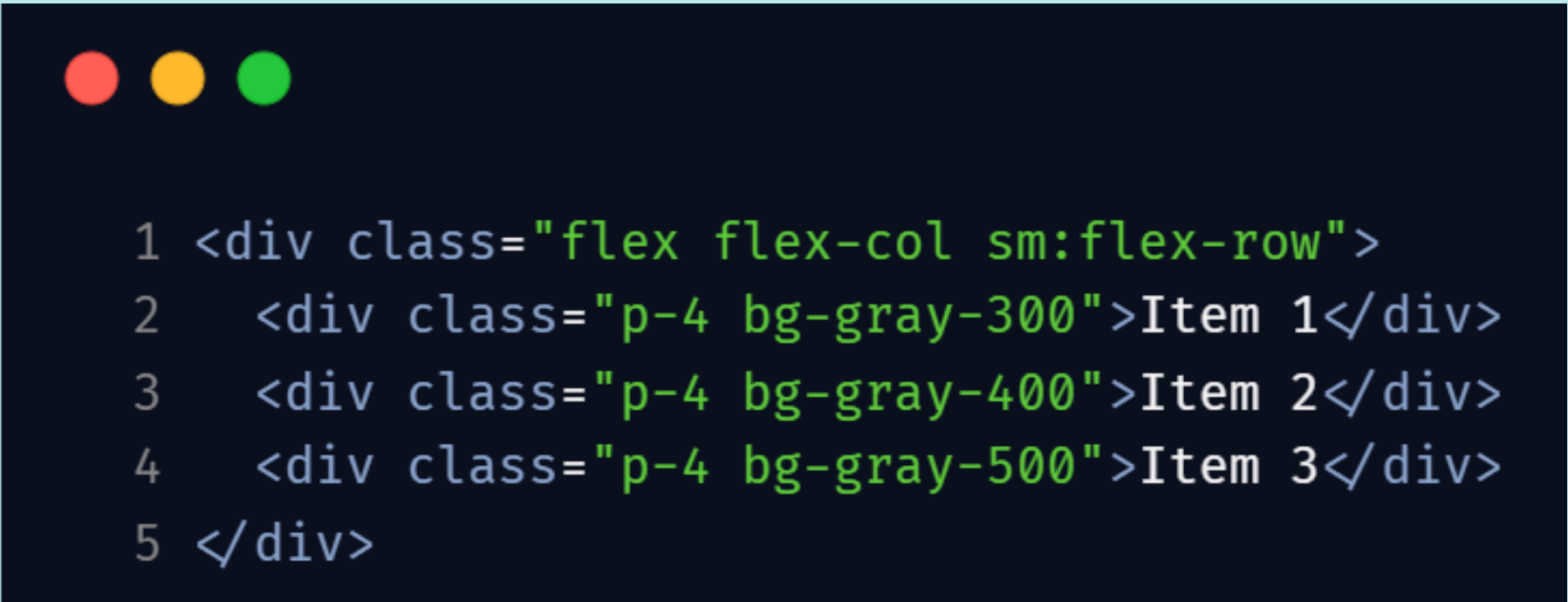
```

Responsive font sizes for different screen sizes

What this code does:

- **text-sm**: Font size for small screens (default).
- **sm:text-base**: Changes to base size on small screens ( $\geq 640\text{px}$ ).
- **md:text-lg**: Larger text on medium screens ( $\geq 768\text{px}$ ).
- **lg:text-xl**: Even larger on large screens ( $\geq 1024\text{px}$ ).

**Responsive Layouts with Flexbox:** Control the layout behavior based on screen sizes. Example:



```
1 <div class="flex flex-col sm:flex-row">
2   <div class="p-4 bg-gray-300">Item 1</div>
3   <div class="p-4 bg-gray-400">Item 2</div>
4   <div class="p-4 bg-gray-500">Item 3</div>
5 </div>
```


- **flex-col**: Stacks items **vertically** on smaller screens.
- **sm:flex-row**: Arranges items **horizontally** for small screens ( $\geq 640\text{px}$ ).



# Responsive Layouts with Grid:

Tailwind's grid utilities make creating responsive grids straightforward.

Example:



```
1 <div class="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-4">
2   <div class="bg-gray-200 p-4">1</div>
3   <div class="bg-gray-300 p-4">2</div>
4   <div class="bg-gray-400 p-4">3</div>
5 </div>
```

- **grid-cols-1:** Single-column grid for smaller screens.
- **sm:grid-cols-2:** Two columns on small screens ( $\geq 640\text{px}$ ).
- **md:grid-cols-3:** Three columns on medium screens ( $\geq 768\text{px}$ ).



### 3. Dark Mode Support

**Dark mode** has become a must-have feature for modern web applications, and **TailwindCSS** makes it incredibly easy to implement. You can toggle between light and dark modes with just a single class.

To make this as easy as possible, Tailwind includes a **darkMode: “selector”** variant that lets you style your app differently when dark mode is enabled:

All you need to do is to add the **darkMode** variant in your `tailwindcss.config.js/ts` file and set its value to **“selector”**, which means that any style selector with the prefix of **‘dark’**, the styles will be applied when dark mode is enabled.

**Note:** The **“class”** value in `tailwindcss.config` file is now updated to **“selector”** in the recent version of `tailwindcss`.

Let's see an example:



```
1 <div class="bg-white dark:bg-slate-800 rounded-lg px-6 py-8 ring-1
  ring-slate-900/5 shadow-xl">
2
3 <h3 className="text-slate-900 dark:text-white text-base font-medium
  tracking-tight">
4   Testing Tailwind Dark mode
5 </h3>
6
7 <p className="text-slate-500 dark:text-slate-400 mt-2 text-sm">
8   Dark mode has become a must-have feature for modern web
   applications, and TailwindCSS makes it incredibly easy to
   implement. You can toggle between light and dark modes with just a
   single class.
9 </p>
10 </div>
```

### Testing Tailwind Dark mode

Dark mode has become a must-have feature for modern web applications, and TailwindCSS makes it incredibly easy to implement. You can toggle between light and dark modes with just a single class.

Light Mode



### Testing Tailwind Dark mode

Dark mode has become a must-have feature for modern web applications, and TailwindCSS makes it incredibly easy to implement. You can toggle between light and dark modes with just a single class.

Dark Mode



## 4. Hover, Focus, and Active States

Tailwind CSS provides utility classes to style elements based on their **interactive states**. These states: **hover**, **focus**, and **active**, allow you to customize how elements respond to user interactions like mouse movements, keyboard focus, or clicks.

```
1 <button class="bg-blue-500 active:bg-green-700 hover:bg-blue-700
  focus:border-blue-500 focus:outline-none text-white font-bold py-2
  px-4 rounded">
2   Hover Me
3 </button>
```

- Default button background is blue-500, changes to green-700 while the button is being clicked.
- When in focus, the border-blue-500 style is applied.
- Default button background is blue-500, changes to blue-700 when hovered.

## 5. Line-clamp out of the box

Truncating long blocks of text is a common requirement in web design, and TailwindCSS simplifies this with the **line-clamp** class.

The line-clamp utility in Tailwind CSS is used to limit the number of lines of text displayed in a block element, adding an ellipsis (...) to indicate truncated content.

This is especially useful when working with dynamic text content, such as blog excerpts, card descriptions, or any multi-line content where space is limited.

For example: Imagine you're building a blog platform and want to display article previews in a card layout. You only want to show the first 3 lines of the blog excerpt for uniformity:

```
1 <div>
2   <h3 class="text-xl font-bold mb-2">Introduction to Tailwind CSS</h3>
3   <p class="line-clamp-3">
4     Tailwind CSS is a highly customizable, low-level CSS framework
5     that gives you all of the building blocks you need to build
6     bespoke designs without any annoying opinionated styles you have
7     to fight to override. This utility-first framework lets you focus
8     on development speed and design consistency.
9   </p>
10 </div>
```

Result:

## Introduction to Tailwind CSS

Tailwind CSS is a highly customizable, low-level CSS framework that gives you all of the building blocks you need to build bespoke designs without any annoying...

Ellipses to  
indicate  
truncated texts


## 6. Before and after pseudo-element variants

The **before** and **after** pseudo-elements in Tailwind CSS are used to add **decorative content** or **styles** before or after an element.

These pseudo-elements are perfect for adding visual elements (like icons, decorations, or indicators) without needing extra markup in your **HTML**.

Tailwind provides utility classes to style these pseudo-elements directly by prefixing your utilities with **before:** or **after:**.

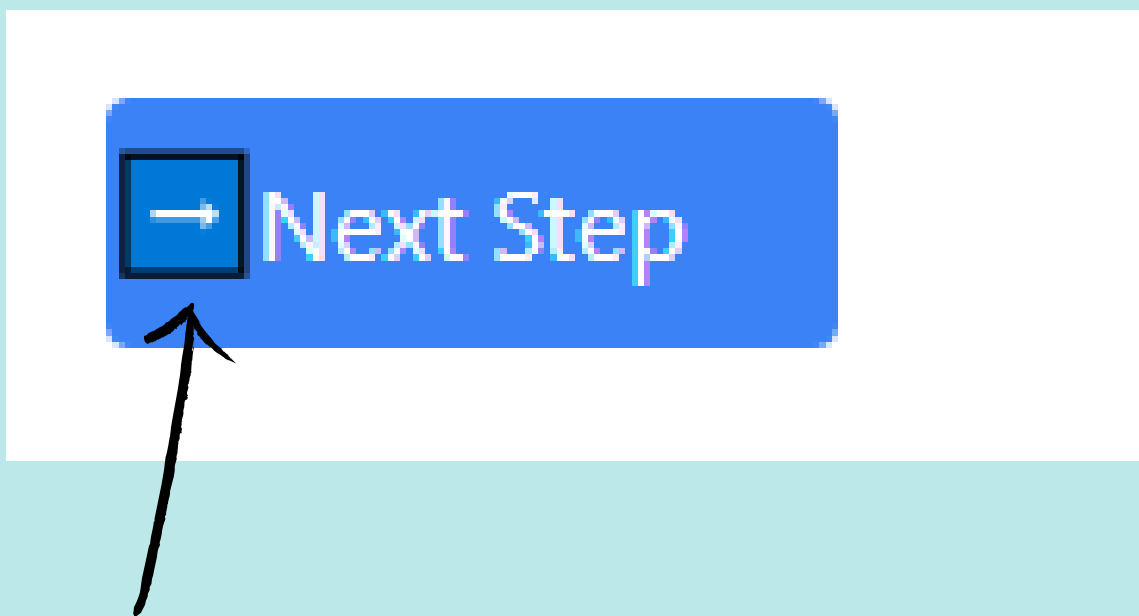
Here's an example of using the **before** pseudo-element to add a decorative arrow before a text button:



```
1 <button class="relative before:content-['➔'] before:absolute  
  before:left-0 before:text-lg before:mr-2 bg-blue-500 text-white py-2  
  px-6 rounded">  
2   Next Step  
3 </button>
```

- **before:content-['➡']**: Adds the arrow emoji before the button text.
- **before:absolute**: Positions the pseudo-element absolutely relative to the button.
- **before:left-0**: Aligns the arrow to the left of the button.
- **before:text-lg**: Sets the font size of the arrow.
- **before:mr-2**: Adds space between the arrow and the button text.

Result:



This arrow was added using the **before:** pseudo element.



Let's use the **after** pseudo-element to add a decorative underline effect:

```
1 <h1 class="relative after:content-[''] after:block after:w-full  
  after:h-1 after:bg-purple-500 after:mt-2">  
2   Stylish Heading  
3 </h1>
```

- `after:content-[""]`: Creates the pseudo-element with an empty string as content.
- `after:block`: Makes the pseudo-element behave like a block element.
- `after:w-full after:h-1`: Sets the width to 100% of the parent and height to 1px.
- `after:bg-purple-500`: Gives the pseudo-element a purple background (acts as the underline).
- `after:mt-2`: Adds some space between the heading text and the underline.

Result:

Stylish Heading

## 7. Accent Color

The **accent-\*** class in Tailwind CSS is used to customize the **accent color** of form controls.

The accent color applies to HTML elements such as radio buttons, checkboxes, and other interactive elements, allowing you to align their appearance with your design.

By default, browsers apply their native accent colors. However, with the **accent-\*** utility, Tailwind enables us to style these elements with consistent branding or a customized look.

# For example: Styling form inputs with accent color

```
1 <form>
2   <label class="flex items-center space-x-2">
3     <input type="checkbox" class="accent-pink-700 h-5 w-5" />
4     <span>Accent color applied</span>
5   </label>
6
7   <label class="flex items-center space-x-2 mt-4">
8     <input type="radio" name="plan" value="basic" class="h-5 w-5" />
9     <span>Browser default</span>
10  </label>
11
12  <label class="flex items-center space-x-2">
13    <input type="radio" name="plan" value="premium" class="accent-red-500 h-5 w-5" />
14    <span>Customized with accent class</span>
15  </label>
16 </form>
17
```

## Result:

- ☒ Accent color applied
- ☒ Browser default
- ☒ Customized with accent class

## 8. Built-in Animations

Tailwind CSS comes with a set of built-in animation utilities that allow you to easily add movement and transitions to your web applications.

These animations are lightweight and customizable, enabling you to create delightful user experiences without additional libraries.

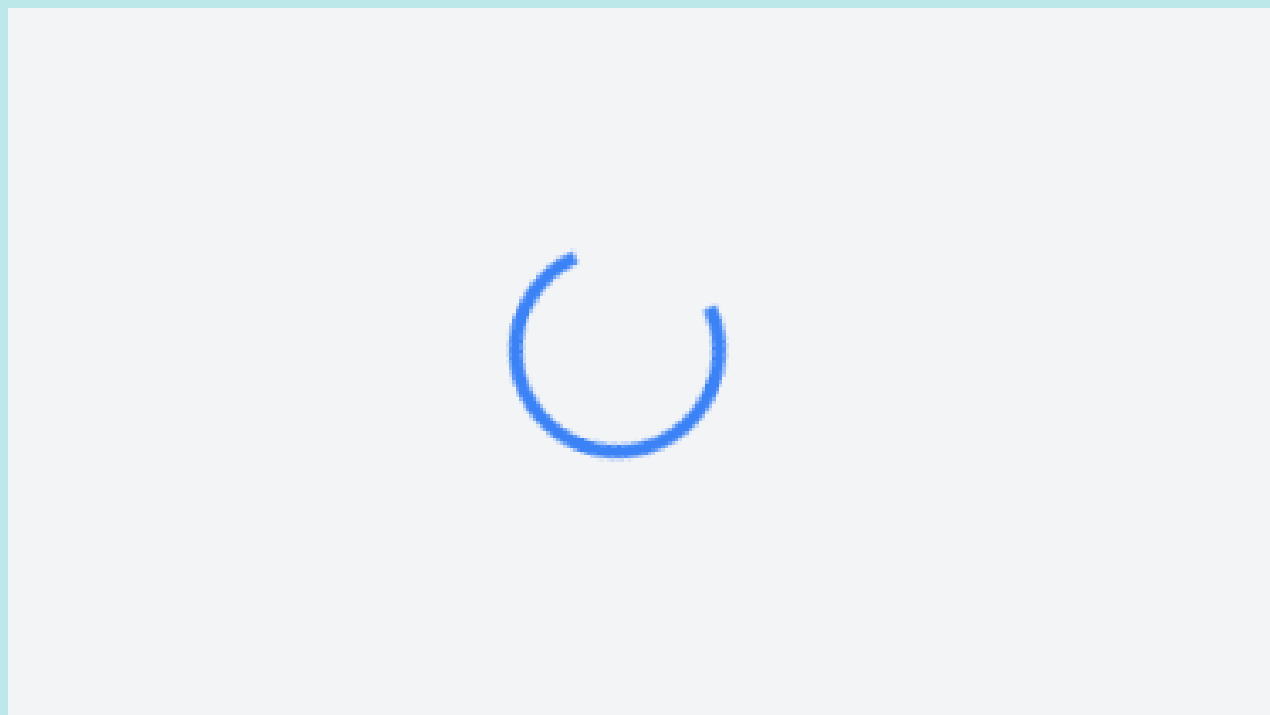
Common Animation Classes:

- **animate-spin**: Rotates an element continuously.
- **animate-ping**: Creates a pulse effect, often used for indicators.
- **animate-pulse**: Smoothly fades in and out.
- **animate-bounce**: Creates a bouncing effect.

# For example: creating a spinning animated loader

```
1 <div class="flex items-center justify-center min-h-screen bg-gray-100">  
2   <div class="animate-spin w-16 h-16 border-4 border-blue-500 border-t-transparent  
   rounded-full">  
3 </div>  
4 </div>
```

Result:



## 9. Layered Utilities for Reusability

Layered utilities in Tailwind CSS allow you to build reusable styles by combining multiple utility classes into custom components.

This feature enhances maintainability and scalability by letting you define common patterns or styles in one place, which can then be applied consistently throughout your application.

Tailwind achieves this reusability primarily using:

1. **@apply Directive (in CSS files)**
2. **Custom Classes**

Tailwind's **@apply** directive lets you group multiple utility classes into a single custom class. This is done in your CSS files, allowing you to create a **shared styling pattern** and this class is used in your **HTML** just like a conventional CSS class.

See how this works in the example below:





```
1 /* styles.css */
2 .btn-primary {
3   @apply bg-blue-500 text-white font-bold py-2 px-4 rounded hover:bg-blue-700
4   focus:outline-none focus:ring-2 focus:ring-blue-300;
5 }
```

The **.btn-primary** class bundles several Tailwind utilities:

- **bg-blue-500**: Blue background color.
- **text-white**: White text.
- **font-bold**: Bold font weight.
- **py-2 px-4**: Padding for vertical and horizontal spacing.
- **rounded**: Rounded corners.
- **hover:bg-blue-700**: Background changes on hover
- **focus:outline-none focus:ring-2 focus:ring-blue-300**: Focus styling for accessibility.

The **.btn-primary**  
class passed as a  
css class




```
1 <button class="btn-primary">
2   Click Me
3 </button>
```

## 10. Arbitrary Value Support

Arbitrary value support in Tailwind CSS allows developers to use custom values directly in utility classes.

This feature is especially useful when you need a specific value that is not included in Tailwind's default configuration or predefined options.

Instead of adding custom values to the Tailwind configuration file, you can define them inline by wrapping the value in square brackets (**[]**).



```
1 <div class="mt-[23px] text-[1.375rem] bg-blue-500 text-white p-4">
2   Custom Margin Example
3 </div>
```

- **mt-[23px]**: Sets a margin-top of exactly 23px.
- **bg-blue-500** and **text-white**: Add a blue background and white text.
- **p-4**: Adds padding of 1rem (from Tailwind's predefined classes).
- **text-[1.375rem]**: Sets the font size to 1.375rem directly.

# 11. Scroll Snap Align

Scroll Snap Align is a feature in Tailwind CSS that allows you to control the alignment of elements within a scrollable container when they snap into view.

This feature is based on the CSS Scroll Snap Module and provides a smooth and controlled scrolling experience, particularly useful for carousels, sliders, or horizontally scrolling menus.

This feature is based on the CSS Scroll Snap Module and provides a smooth and controlled scrolling experience, particularly useful for carousels, sliders, or horizontally scrolling menus.

The key utility classes are:

- **snap-start**: Aligns the start of the element to the container's snap point.
- **snap-center**: Aligns the center of the element to the container's snap point.
- **snap-end**: Aligns the end of the element to the container's snap point.

These classes are used in conjunction with **snap-mandatory** or **snap-proximity** applied to the container to enable snapping.

This feature is based on the CSS Scroll Snap Module and provides a smooth and controlled scrolling experience, particularly useful for carousels, sliders, or horizontally scrolling menus.

These classes are used in conjunction with **snap-mandatory** or **snap-proximity** applied to the container to enable snapping.

As you scroll horizontally, the **snap-center** ensures that the center of each child element aligns with the container's snap point.

The **snap-mandatory** ensures snapping happens after each scroll motion

For example:

```
1 <div class="overflow-x-auto snap-x snap-mandatory flex space-x-4">
2   <div class="snap-center bg-blue-500 w-60 h-40 rounded-lg flex items-center
3     justify-center text-white text-lg">
4     Item 1
5   </div>
6   <div class="snap-center bg-green-500 w-60 h-40 rounded-lg flex items-center
7     justify-center text-white text-lg">
8     Item 2
9   </div>
10  <div class="snap-center bg-red-500 w-60 h-40 rounded-lg flex items-center
11    justify-center text-white text-lg">
12    Item 3
13  </div>
14</div>
```

- **overflow-x-auto**: Enables horizontal scrolling within the container.
- **snap-x**: Activates horizontal snap behavior.
- **snap-mandatory**: Ensures elements always snap to a defined alignment.
- **snap-center**: Aligns the center of each child element to the container's snapping points.

## 12. Simplified RTL support with logical properties

TailwindCSS supports **Right-to-Left (RTL)** languages effortlessly using logical properties.

**Logical** properties (**part of CSS that define layouts based on flow-relative directions instead of physical directions**) ensure styles automatically adapt to the text direction (LTR or RTL) without requiring separate stylesheets or extensive overrides.

For example:

- **margin-left** becomes **margin-inline-start**
- **padding-right** becomes **padding-inline-end**.

Tailwind includes utilities like **ltr** and **rtl** variants and logical classes such as **ms-\* (margin-start)** and **me-\* (margin-end)** to manage these flows.



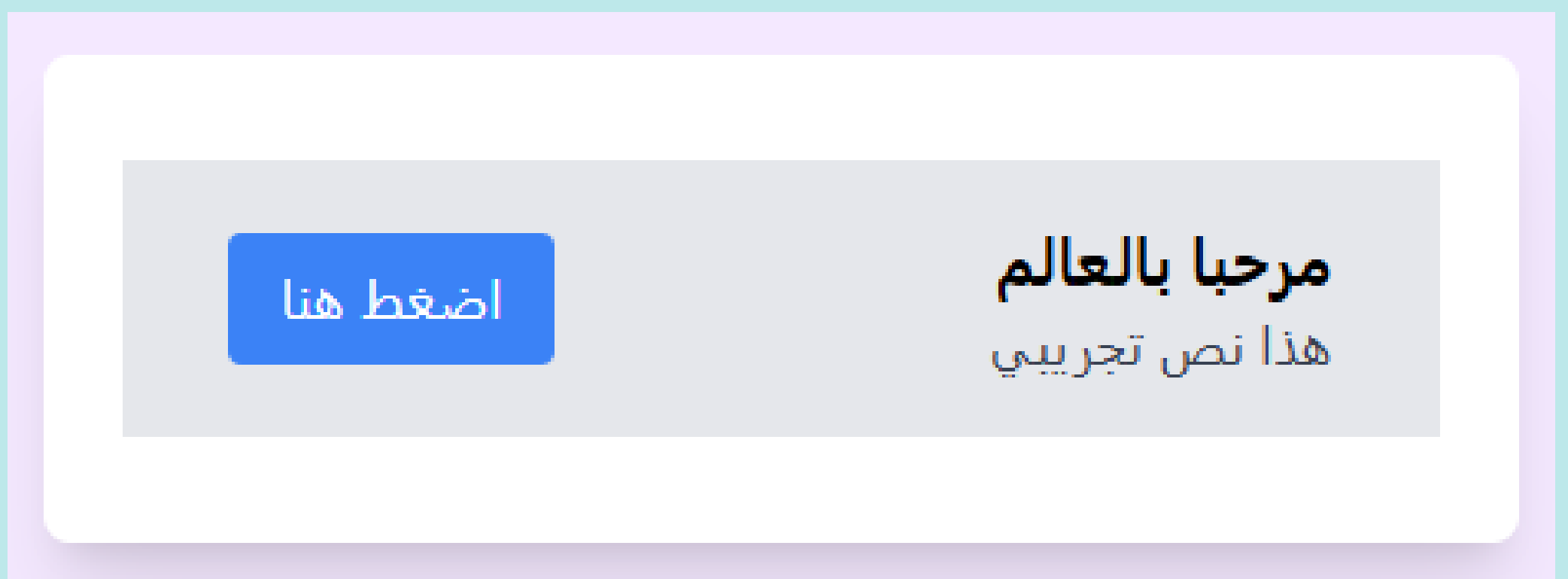
## Example: Responsive RTL Support



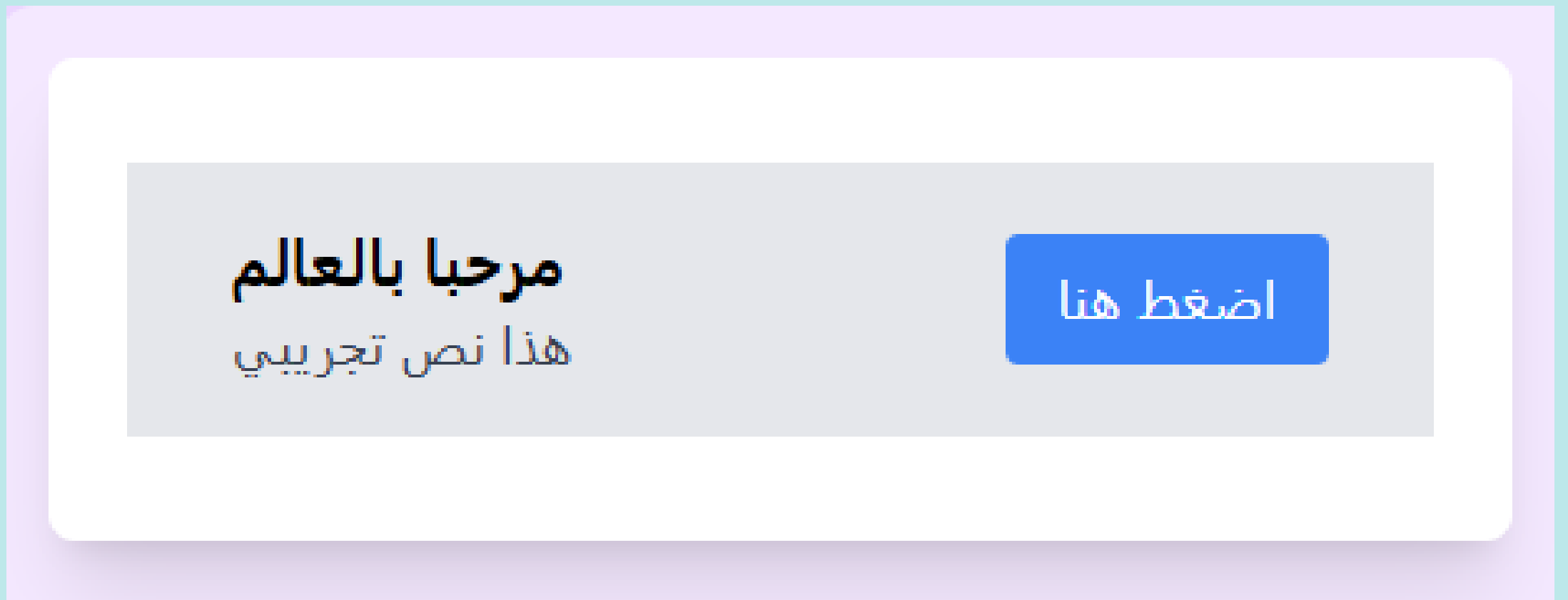
```
1 <div dir="rtl" class="flex justify-between items-center p-4 bg-gray-200">
2   <div class="text-start ms-4">
3     <h1 class="text-xl font-bold">مرحبا بالعالم</h1>
4     <p class="text-gray-700">هذا نص تجريبي</p>
5   </div>
6   <button class="bg-blue-500 text-white py-2 px-4 rounded me-4 hover:bg-blue-700">
7     اضغط هنا
8   </button>
9 </div>
```

- **dir="rtl"**: Sets the direction of the container to RTL.
- **ms-4 (Margin Start)**: Adds a margin to the logical start, which in RTL corresponds to the right side.
- **me-4 (Margin End)**: Adds a margin to the logical end, which in RTL corresponds to the left side.

Result:




And when the '**dir**' value is changed to "**ltr**", it changes the flow direction from left to right alignment like this:



If your application supports multiple languages, especially those requiring LTR, Tailwind's LTR utilities with logical properties allow you to build responsive, multilingual layouts efficiently. 🌍

Here are the 12 features I find really interesting using Tailwind CSS framework for styling my applications.

There are several interesting features in Tailwind CSS, to learn more features, visit tailwind's official documentation here 

<https://tailwindcss.com/docs>

I hope you found this material  
useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be  
helpful to someone 🙌

# Hi There!

## Thank you for reading through

Did you enjoy this knowledge?



Follow my LinkedIn page for more work-life balancing and Coding tips.



LinkedIn: Oluwakemi Oluwadahunsi

[kodemaven-portfolio.vercel.app](https://kodemaven-portfolio.vercel.app)