

# CSS Flex & Grid

---

Complete Guide with  
Real World Examples and  
Code Snippets

Tailwind CSS

SHRUTI BALASA

*"Don't just learn all the things CSS **flexbox** and **grid** can do for you.  
Instead learn all the things YOU can do with them."*

# Complete Guide to CSS Flex & Grid - Tailwind CSS

Version 1.0

Published Online: November 5, 2021

Copyright © 2021 by Shruti Balasa

All rights reserved. No part of this eBook may be reproduced, distributed, or transmitted in any form or by any means, including recording, or other electronic or mechanical methods, without the prior permission of the author, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

For permission requests, send an email to the author at [contact@shrutibalasa.com](mailto:contact@shrutibalasa.com)

## **Disclaimer**

While every effort has been made by the author to present accurate and up to date information within this document, it is apparent technologies rapidly change. Therefore, the author reserves the right to update the contents and information provided herein as these changes progress. The author takes no responsibility for any errors or omissions if such discrepancies exist within this document.

The author accepts no responsibility for any consequential actions taken, whether monetary, legal, or otherwise, by any and all readers of the materials provided.

Readers' results will vary based on their individual perception of the contents herein, and thus no guarantees can be made accurately. Therefore, no guarantees are made.

## About the Author



**Shruti Balasa** is a full stack web developer and a tech educator. In the first six years of her career, she worked at a start-up developing 200+ websites starting from static ones to full-fledged social networking sites and eCommerce websites.

In the past two years, she started sharing her knowledge by creating courses on various platforms, video tutorials on Youtube and through tech talks.

[Follow her on Twitter](#) or visit [her website](#)

# Table of Contents

## Introduction ..... 13

Who is this book for?

How to use this book?

## Why Flex & Grid ..... 18

## Display Flex ..... 21

Example 1a : Quotes Side-by-Side

Understanding `display : flex`

## Justify Content ..... 24

Example 2a : Tabs Spaced Out

Understanding `justify-content`

Example 2b : Card with Previous & Next Links

Example 2c : Team Profiles

## Flex Wrap ..... 29

Example 3a : Responsive Team Profiles

Understanding `flex-wrap`

Example 3b : Logos Wrapped

## Align Items ..... 32

Example 4a : Icon and Text

Understanding `align-items`

Example 4b : Profile Card - Small

Example 4c : Services Section

Example 4d : Frequent Questions

Example 4e : Center a div

**Flex Direction ..... 40**

Example 5a : Welcome Screen

Understanding `flex-direction`

Main Axis and Cross Axis

Example 5b : Testimonial Card

Example 5c : Alternating List of Profiles

**Flex Grow ..... 48**

Example 6a : Inline Subscribe Form

Understanding `flex-grow`

Example 6b : Sticky Footer

Example 6c : Card with Header & Footer

Example 6d : Tabs Hover Effect

Example 6e : Variable Width Responsive Buttons

**Flex Shrink ..... 57**

Example 7a : Itinerary

Understanding `flex-shrink`

Example 7b : Profile Card - Large

**Flex Basis ..... 61**

Example 8a : Split Screen Display

Understanding `flex-basis`

Example 8b : Blog Post Display

Example 8c : Pricing Plans

**Flex Shorthand Property ..... 70**

Understanding `flex`

Example 9a : Navigation Bar with Centered Menu

Example 9b : Image and Text in 2:1 Ratio

**Auto Margins ..... 77**

Example 10a : Notifications Menu Item

Example 10b : Footer with Multiple Columns

**Order ..... 80**

Example 11a : Responsive Navigation Bar

Understanding `order`

**Align Self ..... 83**

Example 12a : Product Display

Understanding `align-self`

Example 12b : Profile with Rating

**Align Content ..... 87**

Example 13a : Full Page Testimonials Section

Understanding `align-content`

**Inline Flex ..... 90**



Example 14a : Social Media Icons

Understanding `inline-flex`

**Comprehensive Examples for Flexbox ..... 94**

Example 15a : Article Preview

Example 15b : Fitness Report

Example 15c : Tweet

**Display Grid & Grid Template Columns ..... 98**

Example 16a : Full Page Gallery

Understanding `display: grid`

Understanding `grid-template-columns`

Example 16b : Layout with Sidebar

Example 16c : Services Grid

Example 16d : Quick Bites Menu

**Grid Template Rows ..... 108**

Example 17a : Sticky Footer with Grid

Understanding `grid-template-rows`

**Gap ..... 111**

Example 18a : Pricing Plans with Grid

Understanding `column-gap`

Example 18b : Blog Posts Display

Understanding `row-gap`

Understanding `gap`

<b>Justify Content</b> .....	<b>116</b>
Example 19a : Featured Logos in a Grid	
Understanding <code>justify-content</code> in Grid	
Example 19b : Shopping Cart Summary	
<b>Align Content</b> .....	<b>122</b>
Example 20a : Profile Card with Bio & Link	
Understanding <code>align-content</code> in Grid	
Example 20b : Features Logos Center of Page	
Understanding <code>place-content</code> in Grid	
<b>Justify Items</b> .....	<b>128</b>
Example 21a : Featured Logos of Different Widths	
Understanding <code>justify-items</code>	
Example 21b : Profile Card with Bio & Link Centered	
<b>Align Items</b> .....	<b>132</b>
Example 22a : Image and Text Section	
Understanding <code>align-items</code> in Grid	
Example 22b : Featured Logos of Different Heights	
<b>Place Items</b> .....	<b>137</b>
Example 23a : Center a div using Grid	
Understanding <code>place-items</code>	
<b>Grid Column Start, End &amp; Span</b> .....	<b>139</b>
Example 24a : Horizontal Form	

Understanding `grid-column-start`

Example 24b : Single Price Grid Component

Understanding `grid-column-end`

Understanding `grid-column`

Example 24c : Page Layout with Grid

**Grid Row ..... 147**

Example 25a : Contact Form

Understanding `grid-row-start` & `grid-row-end`

Understanding `grid-row`

Example 25b : Responsive Services Section

Example 25c : Testimonials Grid Section

**Order ..... 154**

Example 26a : Responsive Pricing Plans

Understanding `order` in Grid

**Advanced Grid Template Values ..... 156**

Example 27a : Pricing Plans with Size Limits

Understanding `minmax()`

Example 27b : Blog Post Page with Code Snippet

Example 27c : Responsive Grid without Media Queries

Understanding `auto-fit`

Understanding `auto-fill`

**Grid Auto Flow ..... 165**

Example 28a : Analytics Section

Understanding `grid-auto-flow`

**Justify Self & Align Self ..... 168**

Example 29a : Restaurant Cards with Labels

Understanding `justify-self` & `align-self`

Example 29b : Caption at the Bottom of Image

**Comprehensive Examples for Grid & Flexbox ..... 172**

Example 30a : Services Section

Example 30b : Twitter Monthly Summary

Example 30c : Social Media Dashboard

**Conclusion ..... 175**

# Introduction

CSS flexbox and grid have become two of the most important topics of web design. Most of the tutorials on the web teach these concepts using some coloured blocks. You get introduced to all the CSS properties related to these concepts and how they work. But very rarely you get to see some examples of where and how these are used in the real world. Without understanding the real world application, learning is incomplete.

## Time for another approach

This book takes a completely different approach. I won't teach you the things flex and grid can do. Instead, I will first show you some components and layouts and make you think how to build them using the Tailwind CSS utility classes you already know. Now you have a problem, and you want a solution. That's when I introduce the concepts you "need" to know.

This is called **Problem-Based Learning (PBL)** which will not only keep you motivated throughout the book, but also help you retain the knowledge far better.

*Shall we get started?*

## Who is this book for?

Whether you are a beginner at Tailwind CSS or CSS itself who has never heard of flex and grid, or someone who knows all the concepts but finding it hard to implement in real projects or somewhere in between, this book is for you. Even if you're here just to look at some examples and practise your Tailwind skills, you will find a great collection here.

## Prerequisites

Throughout this book I will assume that you know the basic concepts of CSS and how to use Tailwind CSS. You need not be great at it, but you need to know some of the basic utility classes for `width`, `height`, `margin`, `padding`, `font`, `color`, `background`, `border`, `position`, `float` and concepts of viewport and responsive web design with Tailwind CSS.

## What not to expect

1. I will not be going through the concepts in the order in which they are usually covered in other tutorials or in the official Tailwind CSS documentation.
2. I will not be talking about installing or setting up Tailwind CSS, configuration, JIT mode and so on.
3. Do not expect to become an expert at these concepts just by reading the book. You need to try out each of the examples, try to think of alternate approaches to get the same output and also think of different similar examples and practise them.

## How to use this book?

I value your purchase and time, so I want to make sure you get the best out of this. Of course, you can skip this section and rush straight to the main content, but I strongly recommend reading this before you jump in:

## Flow of the book

**STEP 1 :** For every new concept, you will first see an example labelled **Example**

**STEP 2 :** You will then see a link **▶ Try it out**

This is a [Tailwind Play](#) link with all the required assets and other styles applied. You can either give it a shot or skip it. I recommend trying it once or at least looking at the *unsolved* output once, so that you'll appreciate and understand the concept I will next present. The examples are such that, it's usually difficult or impossible to get the desired output without the knowledge of the concept I will talk about after that example. So **don't** spend much time on it and **don't** get disappointed if you can't get it working.

**STEP 3 :** I will provide you with the code snippet highlighting the additional utility classes (usually just a couple of them) you can add to the above Tailwind Play link. Then it works! Even without knowing the concept, just looking at those class names, you might be able to make sense of what's happening.

Just in case it didn't work, you can compare your code with the

[▶ Working Demo](#)

**STEP 4 :** Next we will get to explaining the concept and understand the utility classes we just used, and also look at other utilities related to the same concept. This is labelled

**Concept**

**STEP 5 :** You might see some more examples next, to practice the concept that you just learned with different classes related to the same CSS property. Each of these examples have working demo links below them.

**STEP 6 :** And then the cycle continues with new examples and concepts.

The Examples labeled **1a, 1b, 1c** and so on are related to Concept **1**

## Newbie's Guide

If you are completely new to Flexbox and Grid, don't skip any of the steps above. Go through the examples multiple times if needed, until you understand what's going on. Please note that the order in which the concepts are covered in this book is very different from most of the tutorials you will see. So I recommend completing this book fully before you look into other resources online, to avoid confusion.

## Intermediate's Guide

If you have a little knowledge of the Flex and Grid related to Tailwind CSS classes already, you can try out each of the examples and directly compare with the working demo. Even if you got it right, I recommend reading the concept once to reinforce the knowledge you already have.

## Tailwind CSS Version Used

As of releasing this book, the latest version of Tailwind CSS available is **v2.2.15**. I will be referring to the available utilities of this version.

If you are using an older version, please do check for availability of the utility classes used in the examples.

If you are using a newer version, you should not have a problem, unless some utilities get deprecated. I will try and keep this book updated with the addition of utility classes as and when a new version of Tailwind CSS is released. Do check back for an updated version.

Also, all the examples use *JIT mode*.

## Tailwind Play Links

1. The examples require a lot of styling with colors, fonts, spacing, width and more. Adding all these utility classes in HTML will get in the way of learning the necessary flex and grid utility classes. Hence I have extracted all the general styles using `@apply` directive and added them in the custom CSS tab. So, the only utility classes you will see in HTML are the most important ones.
2. There are some examples where a pure Tailwind solution is not available. In those cases, I have taken one of three approaches:
  1. Generated arbitrary styles using square bracket notation in JIT mode
  2. Customized some utility classes in config file (Check [config tab](#) wherever mentioned)
  3. Added custom styles in CSS. Look for them after the comment `/* Important`



styles \*/ (Check [CSS tab](#) wherever mentioned)

3. Each of these links are private. Kindly do not share these links individually anywhere else.

## Reach Out

Feel free to send a mail to [contact@shrutibalasa.com](mailto:contact@shrutibalasa.com) or send a Direct Message on Twitter - [@shrutibalasa](https://twitter.com/shrutibalasa):

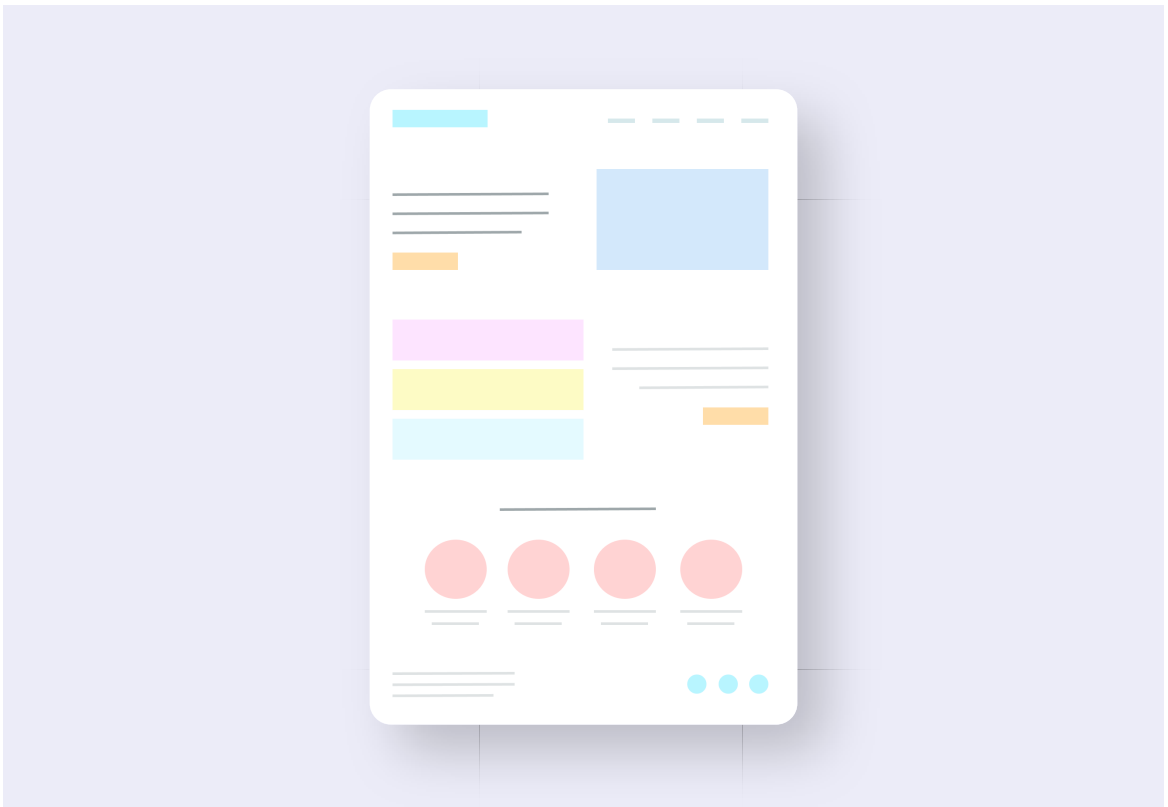
1. If you find any wrong information in this book. I have spent a lot of weeks in research but I could still be wrong. Help me correct the info, so that others don't get misguided.
2. If any of the links are broken or lead to a wrong URL.
3. If you like this book and personally want to let me know how it helped you 😊 Such mails make my day!
4. If you love to talk about this book in your circle, don't do it for free. Reach out to become an affiliate.
5. If you are looking for team pricing.
6. If you want to gift this book to a few people and looking for a discount.

# Why Flex and Grid

Don't you want to first know what problem we are solving?

## The Problem

Any modern web page today looks something like this on a desktop:



Now imagine building this and making it responsive so that it looks great, readable and accessible on smallest of the phones to the largest of the desktops! Assuming you don't know `flex` and `grid`, how would you approach this layout?

## What you might already know

Without any styling, the elements follow the normal flow on the web page. That is, the order in which the elements are specified in your markup is the order in which they usually appear on the web page - one below the other for block-level elements and one next to the other for inline or inline-block level elements. With margins and padding, you can increase or reduce the space between the elements.

Using `relative`, `absolute` or `fixed` positions, you can remove the element from its normal flow and position it elsewhere relative to itself or the page.

With the `float` property, you can make block-level elements appear next to each other but it needs a lot of effort to make full page layouts, like the one above, with just `float`. If you have ever tried it, you know the struggle.

Using `table`, you can achieve the desktop layout easily, but cannot make it responsive.

Now that you understand the problem, let's get to the solution!

## The Solution

Here's presenting the two mighty weapons in CSS - **Flexbox** and **Grid**. You can lay out elements on your web page to build responsive layouts in the best way possible with these. Once you understand and start using these, you will never want to go back to building layouts using any other way!

Let's start with Flexbox first, looking at some examples and master it fully. Then we see what we can do using Grid.

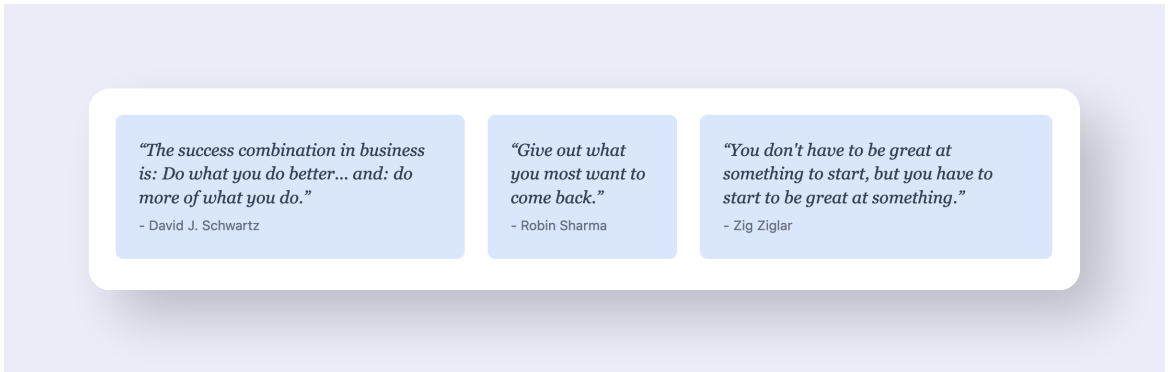
# Flexbox

# 1 Display Flex

Let's look at a very simple example to begin with.

## Quotes Side-by-Side Example 1a

Assume you have three motivational quotes to display on your web page in a single row (on Desktop screen size). You want the blocks to occupy the same height and hence adjust widths based on the length of each quote. These quotes are randomly picked. You don't know how long or short each one is, so you cannot specify widths in fixed units for them.



Here's a Play link for you to try achieving this layout using any of the utility classes you already know:

▶ Try it out

Did you give it a shot? I hope you're convinced that there's no way to achieve this when you don't know how long each quote will be. Can you believe if I tell you this is possible with just **one** utility class? Let's see how.

## Solution

```
1 <div class="flex">
2   <div> ... </div>
3   <div> ... </div>
4   <div> ... </div>
5 </div>
```

You just need to add `flex` class to the parent container. Here's the full working demo.

Tada! 🌟

### ▶ Working Demo

Resize the output panel, rearrange the quotes or add longer ones. Notice how *flexible* the blocks are. This is not yet responsive and you can't add too many quotes yet, but we'll get to those problems soon.

Now that you got a taste of *flexbox*, let's actually understand what it does.

## Understanding Display Flex Concept

**Flexbox** is a method that helps us arrange elements in one direction (horizontally or vertically) and control their dimensions, alignments, order of appearance and more. For this, we need at least two elements - a parent element called **flex container** and at least one child element called **flex item**.

In our above example, the parent element is the flex container, while `.quote` elements are the flex items. And as you just saw, adding `flex` class to any element makes it a flex container.

**Note:** Only the immediate child elements of the container become flex items. Children of flex items are not affected.

Tailwind Class	CSS Property & Value	Explanation
<code>flex</code>	<code>display: flex;</code>	Setting the <code>display</code> property of an element to <code>flex</code> makes it a flex container

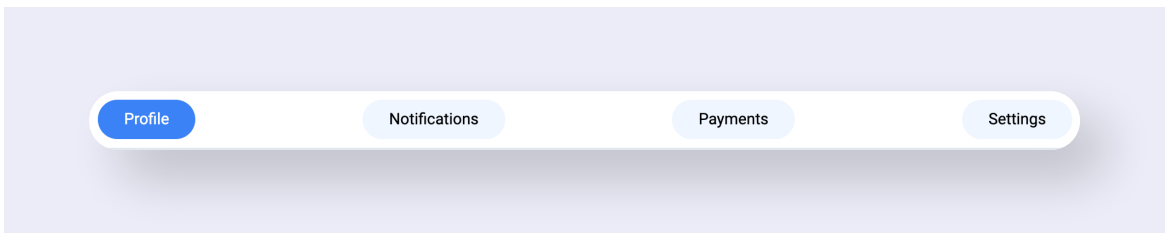
Once you have a flex container and some flex items, there are multiple other Tailwind utilities that can be added to these elements to control the dimensions, alignment, spacing and more. We will be looking at all those classes next, starting with one example for each.

## 2 Justify Content

### Tabs Spaced Out Example 2a

Example contributed by [Naresh](#)

Let's say you have a few tabs on your page and you want them to space out fully with the first tab on the extreme left, last tab on the extreme right and the middle ones spaced out evenly. These tabs have different widths. How would you do it?



You can try this without flexbox if you wish to:

▶ Try it out

I doubt if there is a solution to this without flexbox. Even if you solved this, I'm sure it wasn't an easy approach. Let's see how we can achieve this with flexbox.

### Solution

```
1 <div class="menu flex justify-between">
2   <a> ... </a>
3   <a> ... </a>
4   <a> ... </a>
5   <a> ... </a>
6 </div>
```

▶ Working Demo



Along with the `flex` class, we need to add just one more class `justify-between` to the same element. Let's learn more about these utilities.

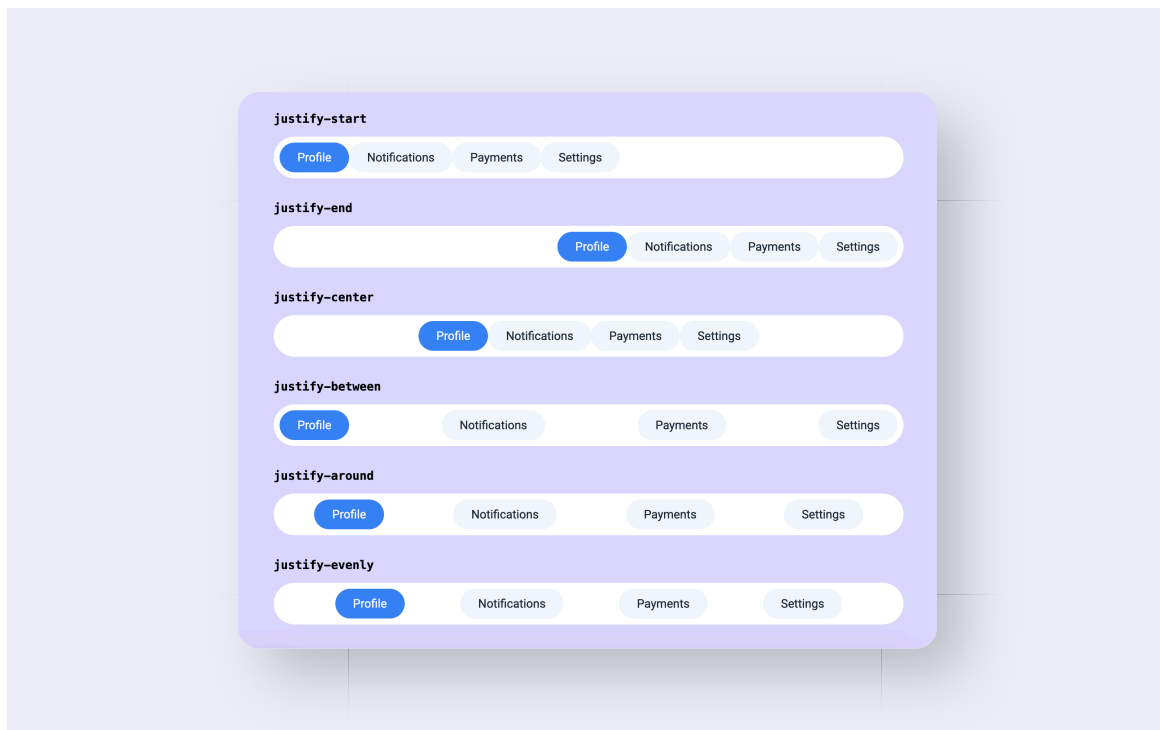
## Understanding Justify Content Concept

Before we understand these utilities, there's something else you need to know. The moment we add a `flex` class to an element, we saw that the children get placed next to each other in one single row. This is a default behaviour. However, we can place them all one below the other in a single column instead. We will get to that a little later.

The utility classes `justify-*` are used to control spacing of the flex items in the direction they are placed. In our above example, it's the horizontal direction. `justify-between` is one of the available utilities we just used. Some more utilities are mentioned below:

Tailwind Class	CSS Property & Value	Explanation
<code>justify-start</code>	<code>justify-content: flex-start;</code>	All items are placed at the beginning of the container with no spaces
<code>justify-end</code>	<code>justify-content: flex-end;</code>	All items are placed at the end of the container with no spaces
<code>justify-center</code>	<code>justify-content: center;</code>	All items are placed at the center with no spaces
<code>justify-between</code>	<code>justify-content: space-between;</code>	All items are spaced out as much as possible with first item at the beginning and last item at the end (We just saw this in action)
<code>justify-around</code>	<code>justify-content: space-around;</code>	Space <i>before</i> the flex items and <i>after</i> the flex items are half as much as space between the items
<code>justify-evenly</code>	<code>justify-content: space-evenly;</code>	Space before, after and between the items are same

You can see the difference between these values below:



Open the working demo, resize the output panel and see how the items move.

[▶ Working Demo](#)

Let's look at some more examples where these utilities would be helpful.

## Card with Previous & Next Links Example 2b

Many times we need two elements at the extreme ends of a section / container, like these "Prev" and "Next" buttons placed at the extreme ends of a card. This is a great example of **flexbox** with `justify-*` utilities used for alignment.

## CSS Flex & Grid

This book takes a completely different approach. I won't teach you the things flex and grid can do. Instead, I will first show you some components and layouts and make you think how to build them using the CSS concepts you already know. Now you have a problem, and you want a solution.

[Prev](#)[Next](#)

Now that you have seen one example, try this out on your own and cross check with the working demo.

[▶ Try it out](#)[▶ Working Demo](#)

## Team Profiles Example 2c

Assume you need to design a "Team" section to display profiles of four people as you can see below. Notice that there is some space to the extreme right and left. This is best achieved with **flexbox** and `justify-around` class for the container.



**Alexa Kardi**  
Founder and CEO



**Tavell Monroe**  
Web Developer



**Adale Smith**  
Marketing Specialist



**Thomas Mason**  
UX Designer

Try it out yourself in the Play link below and then cross check with the working demo.

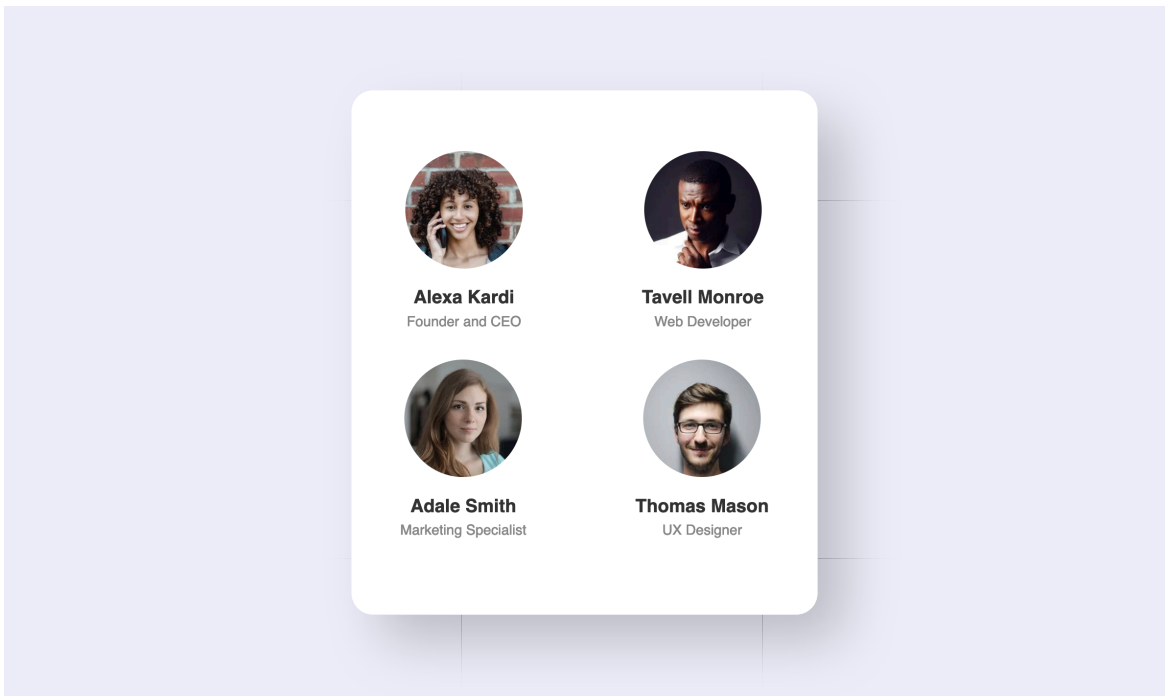
▶ [Try it out](#)

▶ [Working Demo](#)

## 3 Flex Wrap

### Responsive Team Profiles Example 3a

The above examples work great with desktop screen sizes. But try resizing the output panel to a mobile screen size and you will either notice a horizontal scrollbar or the design breaks in some way. How can we make those items move to next row for smaller screens like this?



### Solution

Here's what you can do. Add another class `flex-wrap` to the `container` element:

```
1 <div class="container flex justify-around flex-wrap">
2   <div class="team-profile">
3     ...
4   </div>
5   <!-- Three more team profiles -->
6 </div>
```

▶ Working Demo

## Understanding Flex Wrap Concept

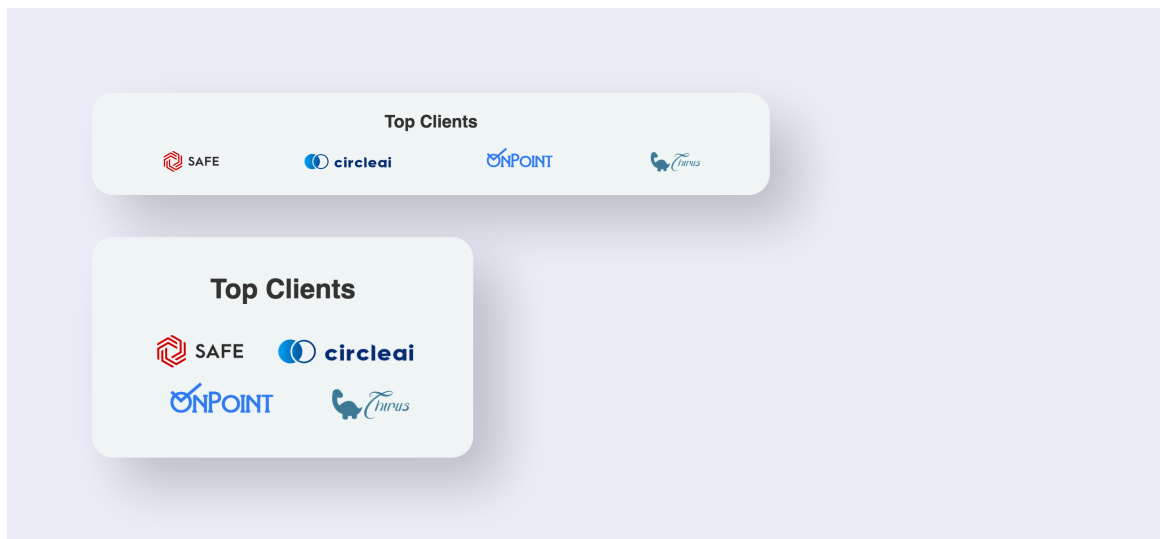
The `flex-wrap` utility class makes the flex items wrap if you run out of space. The default behaviour is to not wrap, which is why the child items do not move into the next row automatically.

Tailwind Class	CSS Property & Value	Explanation
<code>flex-wrap</code>	<code>flex-wrap: wrap;</code>	Items are wrapped into the next line if needed
<code>flex-nowrap</code>	<code>flex-wrap: nowrap;</code>	Items are not wrapped even if it causes overflow
<code>flex-wrap-reverse</code>	<code>flex-wrap: wrap-reverse;</code>	Items are wrapped in the reverse direction

Let's look at another example.

## Logos Wrapped Example 3b

Let's say you need to display a few logos of your clients in a row with *spaces between and around them* and you want them to be responsive on smaller screens. You can use `justify-around` for the spacing and the `flex-wrap` class to wrap the logos.



First three logos contributed by [Gokul](#)

▶ Try it out

### Solution

```
1 <div class="logos flex justify-around flex-wrap">
2   <img ... >
3   <img ... >
4   <img ... >
5   <img ... >
6 </div>
```

▶ Working Demo

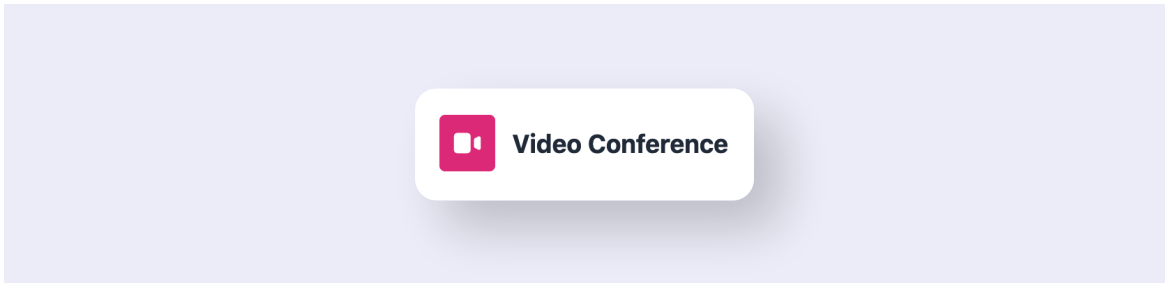
Try out `flex-wrap-reverse` instead, to see the difference.

## 4 Align Items

### Icon and Text Example 4a

Example Credits: [Inovatik](#)

Let's look at another simple use-case of **flexbox**. An icon and text placed next to each other vertically centered.



Without flexbox, can you vertically center align an icon and text like in the above example?

#### ► Try it out

You can try adding `align-middle` class for the `.icon`. But that's not sufficient. You will need to add `align-middle` to the `.icon-text` too. While you might be okay with this adjustment, this is better done with flex.

#### Solution

Instead of the `align-middle` utilities, add these two classes to the `.icon-wrap` element.

```
1 <div class="icon-wrap flex items-center">
2   <span> ... </span>
3   <span> ... </span>
4 </div>
```

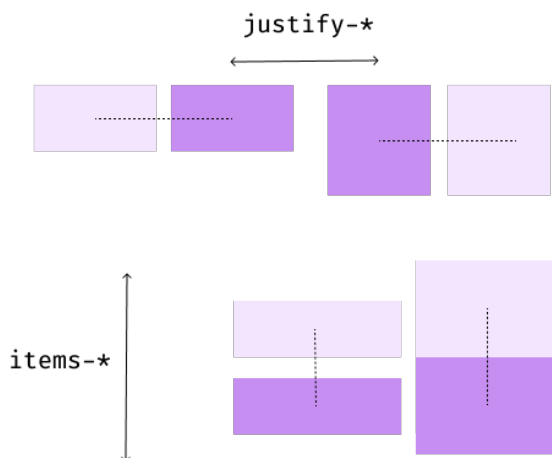
#### ► Working Demo



Apart from `flex`, we added just one more utility - `items-center`. Let's learn more about this.

## Understanding Align Items Concept

The `justify-*` utilities allow us to control the spacing and alignment of the flex items in the direction they are placed (Horizontally in all our previous examples). While `items-*` utilities allow us to control the alignment in its perpendicular direction. This illustration might give you a better idea:

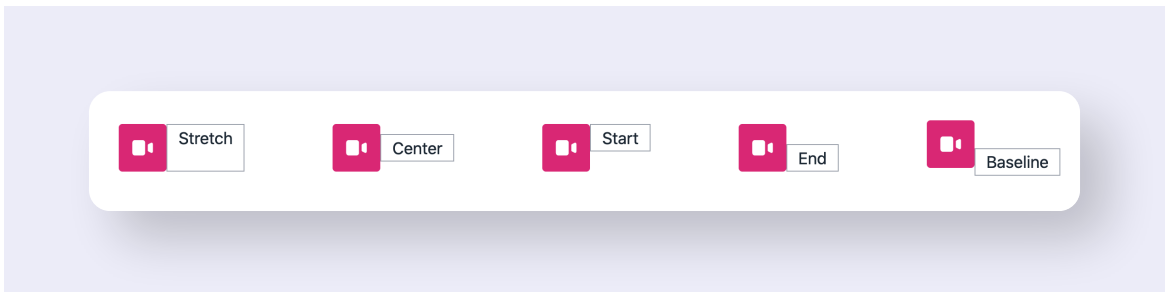


*This illustration is valid only for the concepts we have learned so far. We will talk about these directions again soon*

In case of all our above examples, `justify-*` can be used to space out the items horizontally, and `items-*` can be used to align items vertically. This is useful especially when the height of each item is different. `items-center` is one of the available utilities we just used. Some more utilities are mentioned below:

Tailwind Class	CSS Property & Value	Explanation
<code>items-stretch</code>	<code>align-items: stretch;</code>	All items are stretched to fill the container
<code>items-center</code>	<code>align-items: center;</code>	All items are aligned to the center of the container
<code>items-start</code>	<code>align-items: flex-start;</code>	All items are aligned to the beginning of the container ( <i>at the top in case of the above example</i> )
<code>items-end</code>	<code>align-items: flex-end;</code>	All items are aligned to the end of the container ( <i>at the bottom in case of the above example</i> )
<code>items-baseline</code>	<code>align-items: baseline;</code>	All items are positioned such that the base aligns to the end of the container ( <i>will we talk about this soon</i> )

You can see the difference between these values below:



[▶ Working Demo](#)

To understand the effect of `items-baseline` value, replace the `svg` icon with an alphabet and increase the font size by changing:

```
1 <span class="icon"><svg> ... </svg></span>
```

to

```
1 <span class="icon text-4xl">V</span>
```

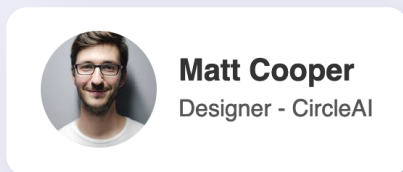


Now you can notice that the base of V is aligned with the base of the word "Baseline", almost like both of them are placed on an invisible line.

The most used utilities are `items-stretch` and `items-center`. So let's look at more of those examples.

## Profile Card - Small Example 4b

Many times we need a component with an avatar and a couple of lines next to it. The `items-center` utility is very useful for such requirements:



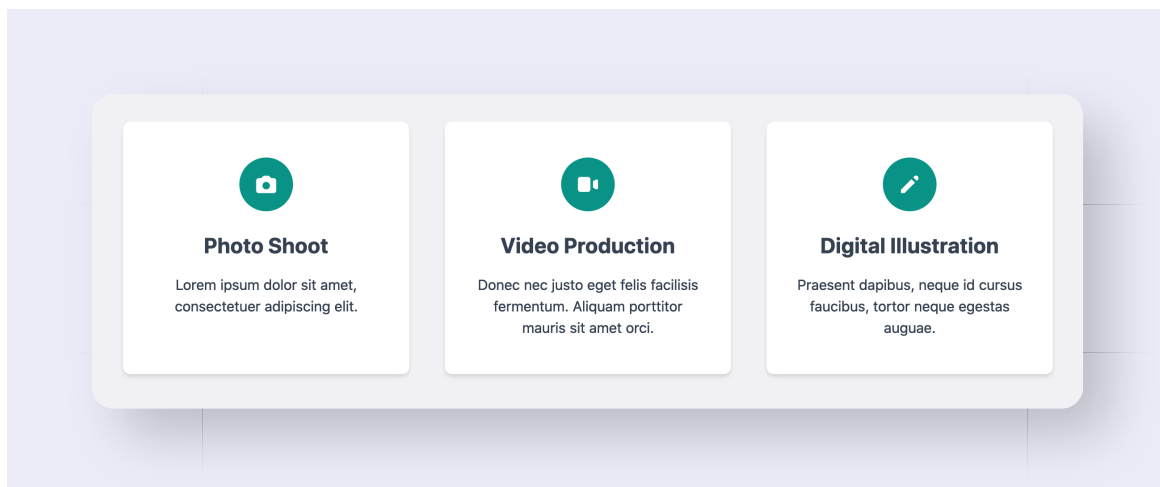
Try doing this yourself before looking at the working code

▶ Try it out

▶ Working Demo

## Services Section Example 4c

When we need to list services as in the below screenshot, the text for one service may occupy 2 lines and for another it may occupy 1 or 3 lines. But we don't want to set a fixed height to keep all the boxes the same height. This is the best use case for the default behaviour of flex items which can also be applied using `items-stretch` utility class.



▶ Working Demo

In the above link, you can remove the `items-stretch` class because its the default. To understand the difference better, change the class to `items-end`

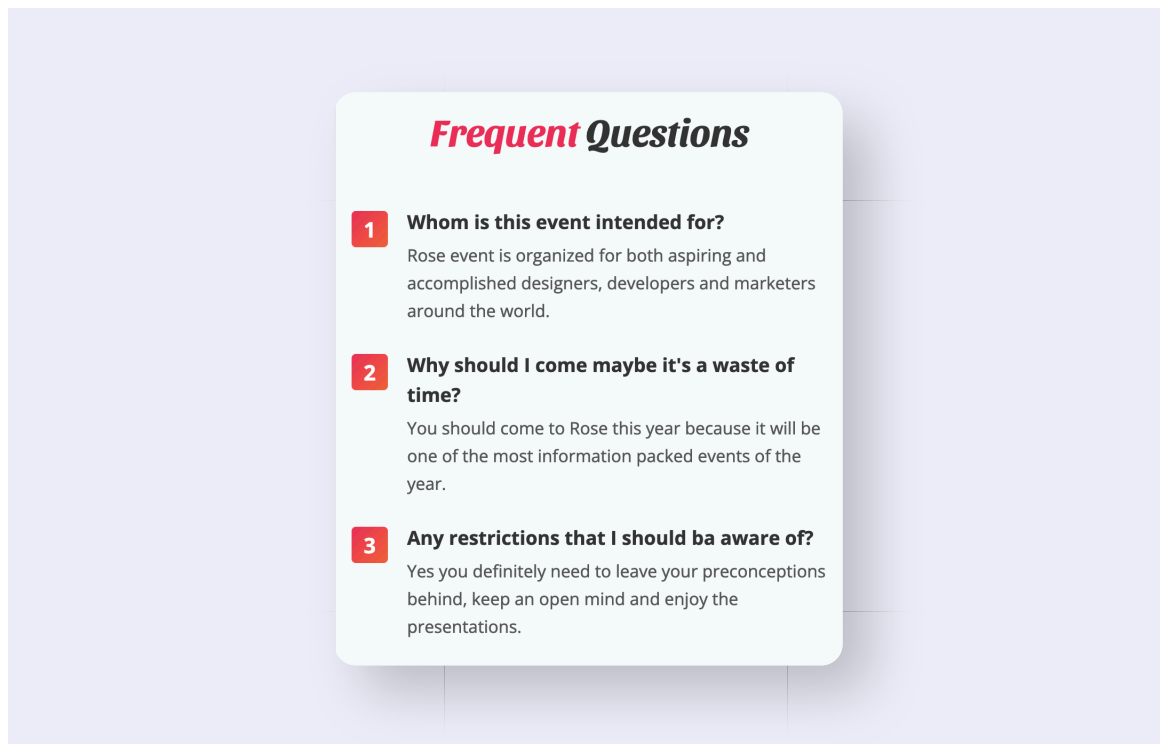
```
1 <div class="container items-end">
```

in the above demo.

## Frequent Questions Example 4d

Example from [Inovatik](#)

Look at this example where some questions are preceded by numbers aligned to the top.



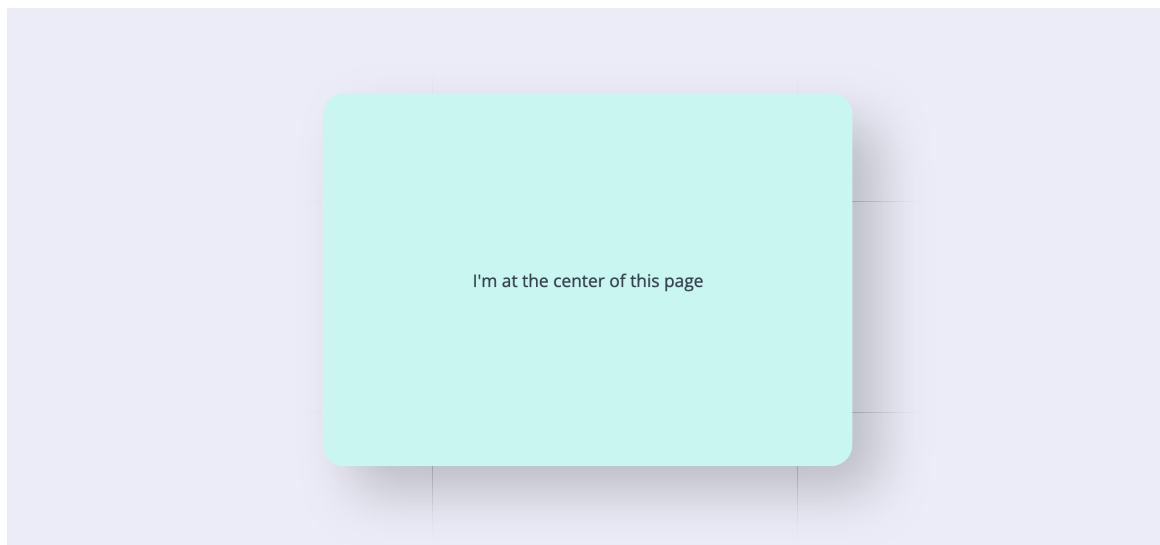
Try using one of the `items-*` utilities to make this happen before looking at the working demo.

▶ Try it out

▶ Working Demo

## Center a div Example 4e

This is something you will always encounter. You want to center a `div` or any element within its parent, but there's no straightforward way to center it both horizontally and vertically. With flexbox, using the `justify-*` and `items-*` utilities, it's super easy.



### Solution 1

We have a container occupying full screen using `w-full` and `h-screen`. Within this, is one `.item` div that we wish to center within the container. Adding `flex` utility to the parent makes it a flex container and the `.item` becomes a flex child. In all the previous examples, we always used more than one flex item. But in this example we need only one.

```
1 <div class="w-full h-screen flex justify-center items-center">
2   <div class="item"> ... </div>
3 </div>
```

Adding `justify-center` and `items-center` positions the child item at the center of the page horizontally and vertically.

▶ [Working Demo](#)

Try changing the width and height of the parent `div` in the link above to see how the `.item` still remains at the center of the container.

## Solution 2

There's another way you could achieve the same result

```
1 <div class="w-full h-screen flex">
2   <div class="item m-auto">... </div>
3 </div>
```

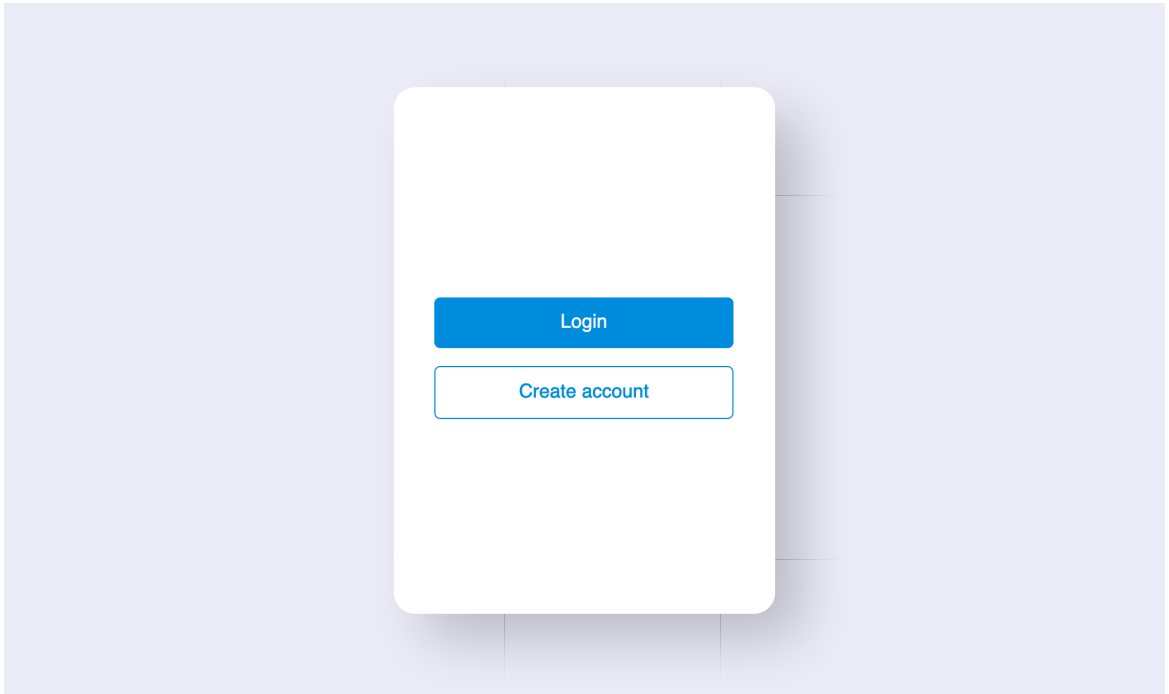
### ▶ Working Demo

Here, instead of using `justify-center` and `items-center` on container, we have used `m-auto` utility on the flex item to set the CSS `margin` property to `auto`. You can use any of the two methods above that suits you.

## 5 Flex Direction

### Welcome Screen Example 5a

Here's an example you will come across a lot. Two or more items vertically centered within its container.



Using the flexbox concepts you just learnt, or without flexbox, can you make this happen?

▶ Try it out



## Possible Solution

There are multiple approaches to this. In case you used the concepts you learned so far, you might have tried adding an additional `div` within wrapper and added `flex` and `items-center` to the wrapper. Additionally adding `block` utilities to the links.

```
1 <div class="wrapper flex items-center">
2   <div class="w-full">
3     <a href="#" class="block link login-link">Login</a>
4     <a href="#" class="block link signup-link">Create account</a>
5   </div>
6 </div>
```

While the above solution works, it's a long one! There's a better approach. You can instead try this:

## Better Solution

Simple add the `flex`, `flex-col` and `justify-center` classes to the parent div:

```
1 <div class="wrapper flex flex-col justify-center">
2   <a href="#" class="link login-link">Login</a>
3   <a href="#" class="link signup-link">Create account</a>
4 </div>
```

### ▶ Working Demo

Confused? You better be 😊 Let's understand what just happened.

# Understanding Flex Direction Concept

The first thing we learnt here was that adding `flex` utility makes all the child elements get laid out in one direction. By default they all get placed in a single row. To change that row direction to a column instead, we can use the `flex-col` utility along with `flex`.

Some more utilities related to the flex direction are:

Tailwind Class	CSS Property & Value	Explanation
<code>flex-row</code>	<code>flex-direction: row;</code>	This is the default behaviour. All items are placed in a single row from left to right
<code>flex-col</code>	<code>flex-direction: column;</code>	All items are placed in a single column from top to bottom
<code>flex-row-reverse</code>	<code>flex-direction: row-reverse;</code>	All items are placed in a single row from <b>right to left</b>
<code>flex-col-reverse</code>	<code>flex-direction: col-reverse;</code>	All items are placed in a single column from <b>bottom to top</b>

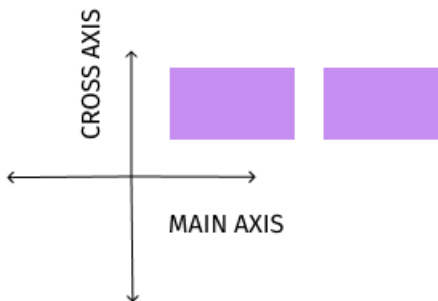
At first, it appears that using `flex-col` with `flex` is same as the normal flow of the web page. Without flexbox, this is how the elements are placed anyway. Then why do we need this? Like we just saw in [Example 5a](#) above, this is the best way to vertically align those two buttons in the center of the container. There are few more use cases of `flex-col` that we will explore further.

Before that, I want you to notice one thing. In the above example, we used `justify-center` to center the items **vertically**. In [Example 4a](#) and [Example 4b](#), we used `items-center` to center the items vertically! Now why is that?

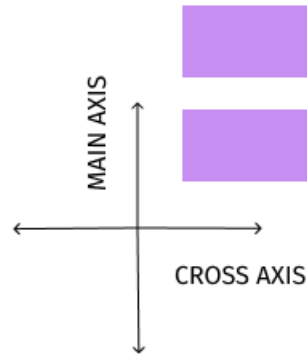
## Main Axis and Cross Axis

When the flex direction is `row`, X axis is the main axis and Y axis is the cross axis. But for flex direction `column`, Y axis is the main axis and X axis is the cross axis.

### row & row-reverse



### column & column-reverse

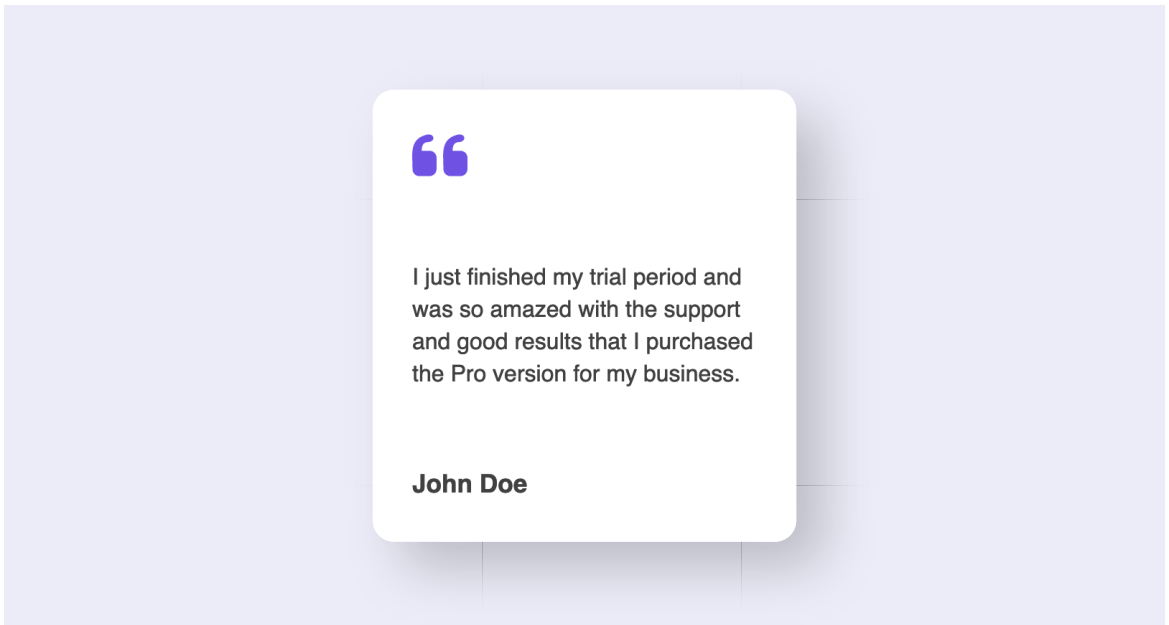


The `justify-*` utilities control spacing and alignment along the **main axis**, while the `items-*` utilities control alignment along the **cross axis**. In Example 5a, our flex direction is `column`. So vertically centering needs alignment along its main axis. That's why we need to use `justify-center`.

This concept requires some practice. Let's look at more examples now.

## Testimonial Card Example 5b

Assume you have a testimonial card with fixed height. Within the card, there's a quote icon at the top, customer name at the bottom and the testimonial text at the middle. The testimonial text can vary in length, but needs to be equally spaced from the icon and the name.



Now use the `flex-col` class along with few more necessary classes and see if you can get the desired result.

▶ Try it out

### Solution

We need to apply 4 utility classes to to the container `.card`

```
1 <div class="card flex flex-col justify-between items-start">
2   <img ... >
3   <p> ... </p>
4   <span> ... </span>
5 </div>
```

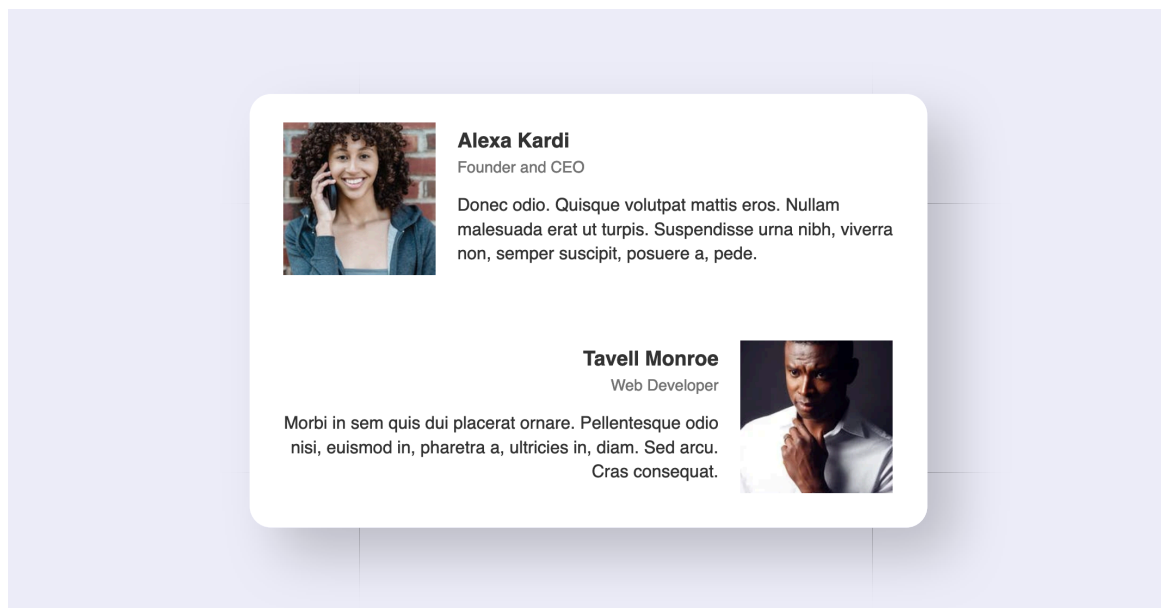
▶ Working Demo

By default, the flex items stretch along the cross axis. So without the `items-start` class, the icon image stretches full width because that's the cross axis when `column` direction is used. The utility class `justify-between` is what makes the child items space out vertically as required.

Try adding more lines to the testimonial text or removing some lines. You will notice that the text still remains equally spaced from the icon and name, as required.

## Alternating List of Profiles Example 5c

Let's say you have to list some profiles on your page. To break the monotony, you'd like to alternate the photos and text like this.



One way is to directly change the order in HTML.

## Markup

```
1 <div class="profile">
2   <img ... >
3   <div> ... </div>
4 </div>
5 <div class="profile">
6   <!-- Reverse the order -->
7   <div> ... </div>
8   <img ... >
9 </div>
```

But if you have a long list and suddenly you wish to insert another profile somewhere in between, you will have to again reverse the order in the markup for all the profiles that appear after that.

Using `flex-row-reverse` only for `even` child items, you can achieve this without changing the order.

▶ Try it out

## Solution

```
1 <div class="profile flex items-center even:flex-row-reverse even:text-
   right">
2   <img ... >
3   <div> ... </div>
4 </div>
```

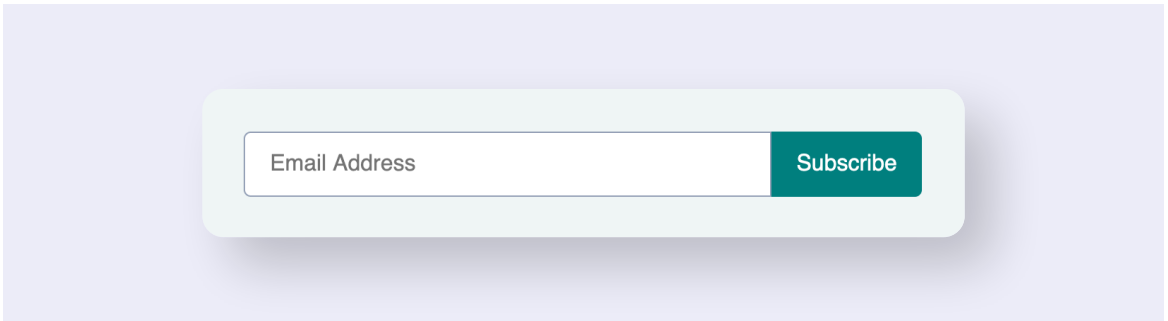
▶ Working Demo

The `even:` prefix helps apply the `row-reverse` direction only to the even child elements. This solution is helpful when you are using frameworks like Vue, React or Laravel where you have the profiles data stored in objects and you display them using loops.

## 6 Flex Grow

### Inline Subscribe Form Example 6a

Here is a subscribe form with a text input and a button displayed in a single row. So flexbox is the best solution, but how do you make the text input occupy all the available horizontal space of its parent container?



Try out and see how you can make the text input occupy the entire space available while the **Subscribe** button takes up only as much space as needed. (*Don't take too long trying because we have a very simple utility class for this*)

▶ Try it out

### Markup

```
1 <div class="container flex">
2   <input ... >
3   <button> ... </button>
4 </div>
```

We already have `flex` applied on `.container`. Now we need to add one class to the `input` element.



## Solution

```
1 <input class="flex-grow" ... >
```

### ▶ Working Demo

This just works! Resize the browser and it's fully responsive.

Notice that until now, we only added classes to the parent element - the **flex container**. The class `flex-grow` is the first one to be used on a child element - the **flex item**. Now let's learn more about these utilities.

## Understanding Flex Grow Concept

The default behaviour of a flex item is to occupy only as much space as needed by the content within. It doesn't "grow", because the default value of the CSS `flex-grow` property is 0. By adding the `flex-grow` class to an element, we are changing the element's `flex-grow` to 1. In CSS, you can set this to any number greater than 0. This value is also called the **grow factor**. You can make the item occupy the left over space (*Left over width in case of row direction, and left over height in case of column direction*).

In the previous example, we added the rule `flex-grow` for the text input. This made the input field occupy all the left over width in the parent. What if we add the same rule to the button as well? Try for yourself in the same demo link above.

Notice how the button also tries to occupy some of the left over horizontal space. Also notice that we added a grow factor of **1** to both the items, but they don't have equal widths. This is where it's easy to get confused. Read the next part carefully.

When `flex-grow` is added to two flex items, the left over space is divided into two parts and added to the **initial** widths of those two items. Since the text input's initial width was more than that of the button, it occupies more space. In Tailwind CSS, we have only two utility classes available with respect to flex grow.

Tailwind Class	CSS Property & Value	Explanation
<code>flex-grow</code>	<code>flex-grow: 1;</code>	The item grows to occupy remaining space along the main axis
<code>flex-grow-0</code>	<code>flex-grow: 0;</code>	This is the default. The item occupies only as much space as needed even if more space is available

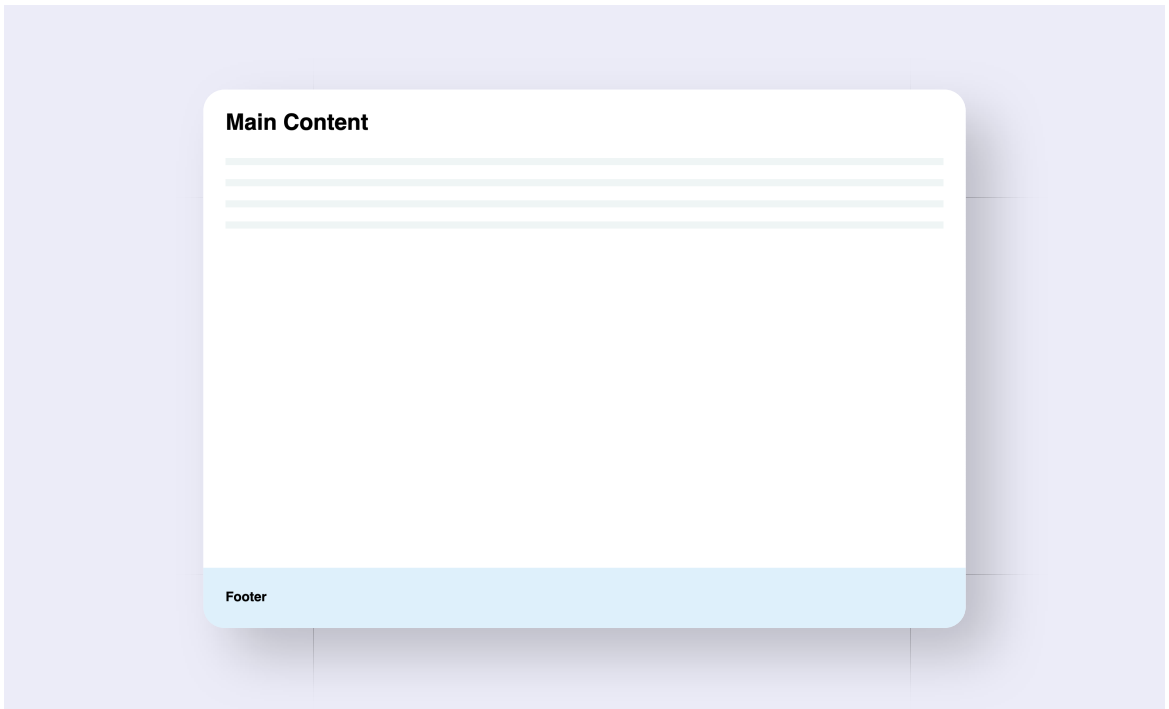
But if we want to set a higher `flex-grow` value, we can add it in the config file or use arbitrary values or add custom styles. What if we add `flex-grow: 2` to the text input, but `flex-grow: 1` to the button? This time, the left over space is divided into **three** equal parts. Two parts width is added to the text input and one part width to the button 🤪

If this is too confusing, just don't worry. In the next few examples, all of this will become clear. For now, just remember:

1. `flex-grow` is a flex item's utility (and not of flex container)
2. It can take any value greater than or equal to 0.
3. The default value is 0, hence the flex item does not grow by default

## Sticky Footer Example 6b

Ever faced a situation where your main content is too small, making your footer appear somewhere in the middle of the page instead of at the bottom? The easiest solution to this is using flexbox for the whole layout, with *column* direction and adding `flex-grow` to the main content.



▶ Try it out

### Markup

```
1 <div class="container">
2   <div class="main"> ... </div>
3   <footer> ... </footer>
4 </div>
```

We need to first add a `min-h-screen` to the `.container`. Otherwise nothing will work. Then make it a flex container with `flex` and `flex-col`. At last, add `flex-grow` utility to the `.main` element.

## Solution

```
1 <div class="container min-h-screen flex flex-col">
2   <div class="main flex-grow"> ... </div>
3   <footer> ... </footer>
4 </div>
```

### ▶ Working Demo

If the main content is long enough, the footer is at the bottom as usual. Which is why it's called a "Sticky Footer". Do check for yourself.

## Card with Header & Footer Example 6c

This one is very similar to our previous example. Let's say we have a card of a specific height - like a blogpost preview with title (as header), an excerpt and a "Read more" button (as footer). The excerpt might sometimes be small, but you would want your button to "stick" to the bottom of the card regardless of the height of the excerpt.

## The Power of CSS Flexbox

Phasellus ultrices nulla quis nibh. Quisque a lectus. Donec consectetur ligula vulputate sem tristique cursus. Nam nulla quam, gravida non, commodo a, sodales sit amet, nisi.

[Read more](#)

Since this is very similar to the previous example, I would encourage you to try it out first before looking at the working demo.

[▶ Try it out](#)

[▶ Working Demo](#)

## Tabs Hover Effect Example 6d

Here's an example of tabs that expand on hover. Each tab has a variable width depending on the text. Once hovered, the active tab expands while the other two shrink.

Description

Care Instructions

Return Policy

Description 

Care Instructions

Return Policy

Can you try and achieve this by changing the `flex-grow` value on hover (using arbitrary styles with `[]`)?

▶ Try it out

## Markup

```
1 <ul>
2   <li> ... </li>
3   <li> ... </li>
4   <li> ... </li>
5 </ul>
```

## Solution

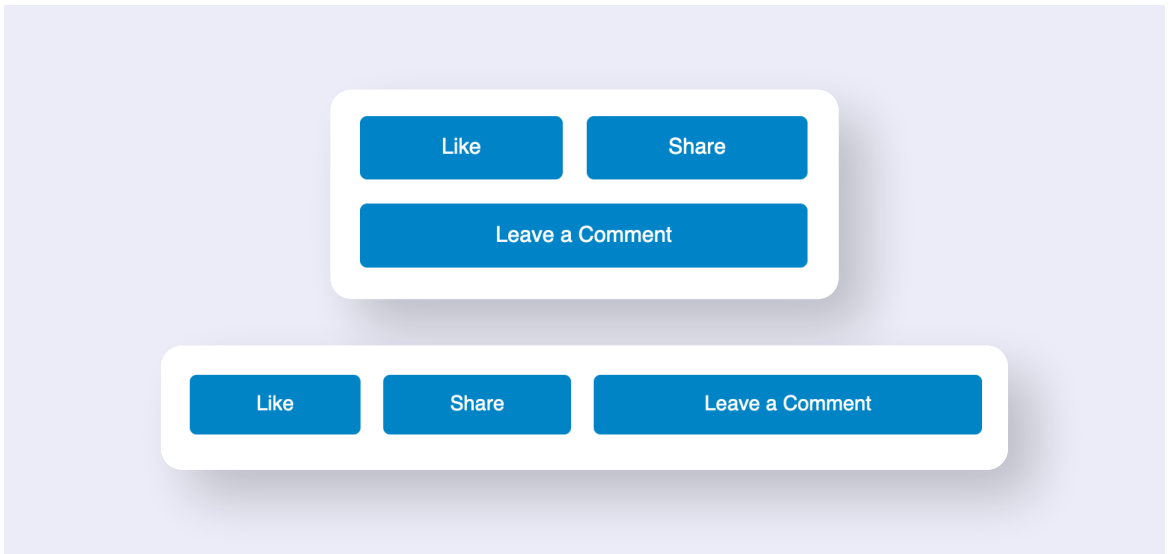
Initially, all the tabs are set to `flex-grow: 1` and on hover, we increase the value of `flex-grow` to any number depending on how wide you want the active tab to be.

```
1 <ul class="flex">
2   <li class="flex-grow hover:flex-grow-[3]"> ... </li>
3   <li class="flex-grow hover:flex-grow-[3]"> ... </li>
4   <li class="flex-grow hover:flex-grow-[3]"> ... </li>
5 </ul>
```

▶ Working Demo

## Variable Width Responsive Buttons Example 6e

Consider this example where you have three buttons below a blog post - "Like", "Share" and "Leave a Comment". You want them to occupy full width of the container and also want to give importance to the last button by giving it a larger width compared to the other two buttons.



And yes, this is very easy with `flex-grow`. Also, when you set the `flex-wrap` property to `wrap`, you get a responsive solution without using any media queries.

▶ Try it out

### Markup

```
1 <div class="container">
2   <button type="button"> ... </button>
3   <button type="button"> ... </button>
4   <button type="button"> ... </button>
5 </div>
```

### Solution

```
1 <div class="container flex flex-wrap">
2   <button type="button flex-grow"> ... </button>
3   <button type="button flex-grow"> ... </button>
4   <button type="button flex-grow-[2]"> ... </button>
5 </div>
```

## ▸ Working Demo

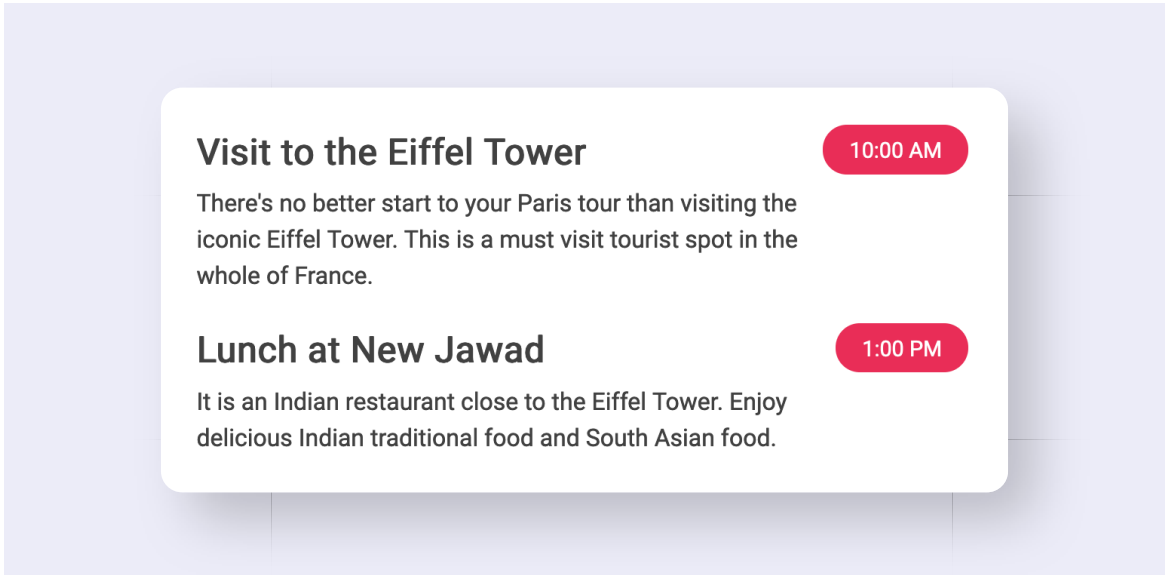
We will look at more examples that use `flex-grow` once we learn a couple more properties.




## 7 Flex Shrink

### Itinerary Example 7a

Here's a simple itinerary component with the description on left and time on right.



It looks simple and easy to achieve using flexbox, but because flexbox decides the width of each child item based on the content within, the time element gets a very little space making it appear in two lines like this 

A HTML solution to this is to wrap the time in `<noabr>` tags. But can you find a CSS solution to this problem?

▶ Try it out

Let's see how to get this working using another set of **flex item's** utility classes `flex-shrink`

## Markup

```
1 <div class="container">
2   <div> ... </div>
3   <span>10:00 AM</span>
4 </div>
```

## Solution

```
1 <div class="container flex items-start">
2   <div> ... </div>
3   <span class="flex-shrink-0">10:00 AM</span>
4 </div>
```

This will prevent the time `span` from *shrinking*.

### ▶ Working Demo

You might immediately see that `flex-shrink` is somewhat opposite to `flex-grow`. Let's learn about these utilities in detail.

## Understanding Flex Shrink Concept

The default behaviour of flex items is to shrink to fit in a single row or a single column of the container (unless `flex-wrap` is set to `wrap`). Hence each item shrinks proportionate to its initial size. You don't have to get into the exact calculations. A larger element shrinks more than the smaller one by default. This is because, the default value of `flex-shrink` for each flex item is **1**.

In our previous example, we changed this value to **0** using the utility class `flex-shrink-0`, hence preventing the item from shrinking. The other item shrinks to fit. The way **grow factor** specifies how much additional space the item should occupy, the **shrink factor** specifies how much space should be reduced from the flex item's initial width.

In Tailwind CSS, we have only two utility classes available with respect to flex shrink.

Tailwind Class	CSS Property & Value	Explanation
<code>flex-shrink</code>	<code>flex-shrink: 1;</code>	This is the default. The item shrinks along the main axis to fit in a single row.
<code>flex-shrink-0</code>	<code>flex-shrink: 0;</code>	The item does not shrink even if it causes the container to overflow

You will mostly never use a shrink factor other than 0 or 1. You would either want the element to shrink or not. So, you only need to remember these:

1. `flex-shrink` is a flex item's property
2. It can take any value greater than or equal to 0.
3. The default value is 1, hence the flex item shrinks by default regardless of the specified `width`.

## Profile Card - Large Example 7b

We saw a small profile card in [Example 4b](#). Since that's small, it's responsive as it is. But if you add a long description instead of small text, the image on the left shrinks to become an oval on smaller screens.



### Matt Cooper

A front end web developer from New York, USA. Currently working as a freelancer. Drop a mail or say hello 🙌

Can you make the image not shrink?

#### ▶ Try it out

Yes, one solution is to change the `width` to `min-width`. It works for this example, but sometimes we might not know the exact width. Hence its best to use `flex-shrink`.

#### Markup

```
1 <div class="profile">
2   <img ... >
3   <div> ... </div>
4 </div>
```

#### Solution

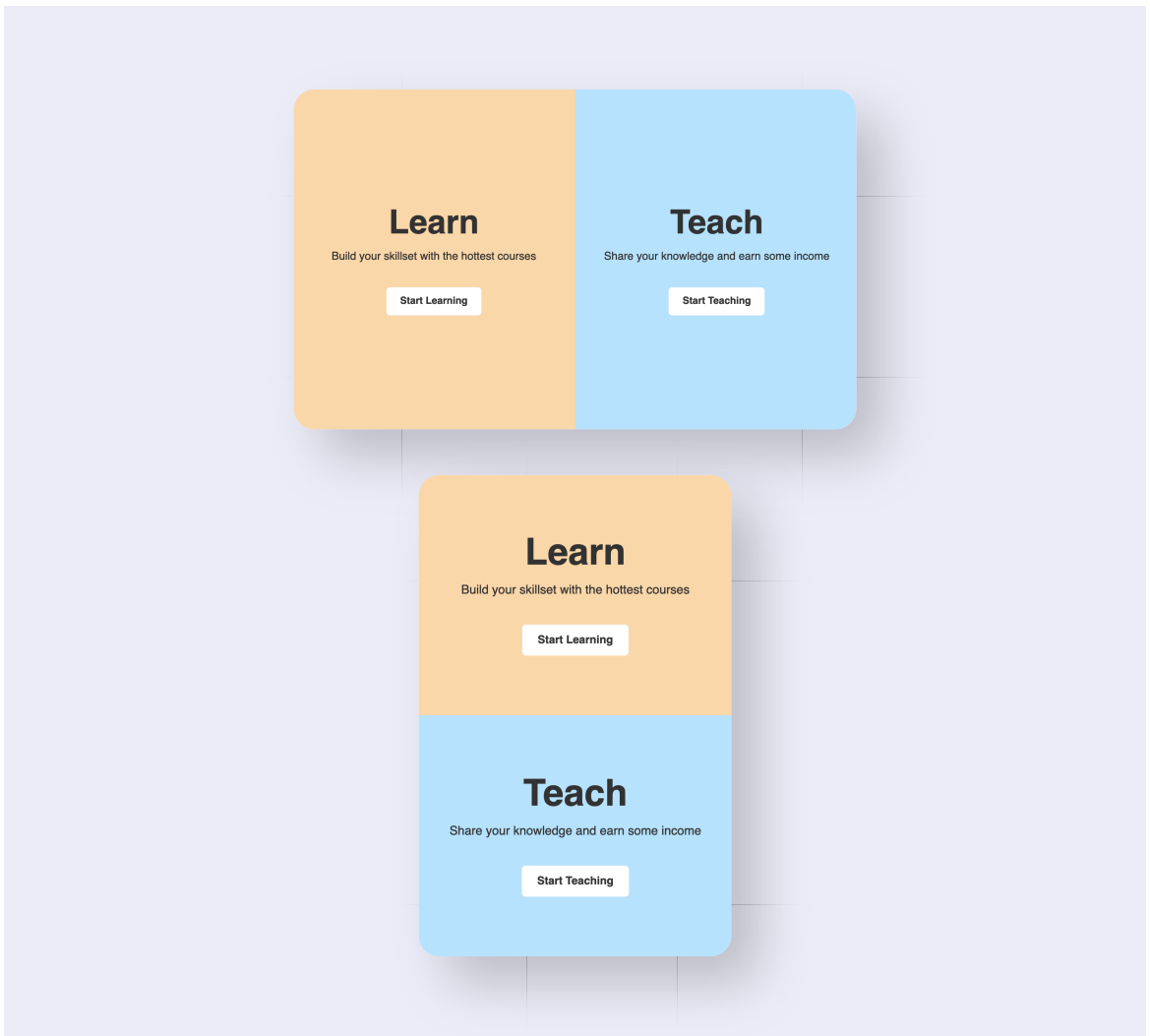
```
1 <div class="profile flex items-center">
2   <img class="flex-shrink-0" ... >
3   <div> ... </div>
4 </div>
```

#### ▶ Working Demo

## 8 Flex Basis

### Split Screen Display Example 8a

Here's a simplified example of a landing page with a split screen display occupying full screen. On large screens, the page is split up horizontally and on smaller screens, it's split up vertically.



With the concepts you've learned so far, I'm sure you can make this work. Since Tailwind CSS uses mobile first approach, you need to first use `flex-col` to split the screen vertically for small screens along with `height` utilities for half height. For wider screens, you can use the `md:` prefix and change to `flex-row` and using `width` utilities, split the screen horizontally.

▶ Try it out

### Markup

```
1 <div class="container">
2   <div class="split"> ... </div>
3   <div class="split"> ... </div>
4 </div>
```

The `.container` element should be full screen. That's done with `w-full` and `h-screen`. Also, the container's flex direction is set to column on smaller screens and row on large screens. So this is achieved with `flex flex-col md:flex-row`.

Then, you can add a `h-1/2` to the flex items on small screens. And on large screens, `w-1/2` and change the height back to `h-full`.

### Possible Solution 1

```
1 <div class="container w-full h-screen flex flex-col md:flex-row">
2   <div class="split h-1/2 md:w-1/2 md:h-full"> ... </div>
3   <div class="split h-1/2 md:w-1/2 md:h-full"> ... </div>
4 </div>
```

Or, you might can add a `flex-grow` to both the flex items.

## Possible Solution 2

```
1 <div class="container w-full h-screen flex flex-col md:flex-row">
2   <div class="split flex-grow"> ... </div>
3   <div class="split flex-grow"> ... </div>
4 </div>
```

Both the solutions work for this example. Solution 1 is long but works all the time. Solution 2 is short but might not work for different cases (Example, one split screen contains a large image).

## Better Solution

```
1 <div class="container w-full h-screen flex flex-col md:flex-row">
2   <div class="split basis-1/2"> ... </div>
3   <div class="split basis-1/2"> ... </div>
4 </div>
```

But `basis-1/2` is not yet a utility class in v2.2.15. It will soon be added to v3+. Until then, we can add custom CSS like this:

```
1 @layer utilities {
2   .basis-1\2 {
3     flex-basis: 50%;
4   }
5 }
```

### ▶ Working Demo

So `flex-basis` property for a flex item is similar to width for `flex-row` direction and similar to height for `flex-col` direction.

# Understanding Flex Basis Concept

The property `flex-basis` is another one that can be defined on the **flex item** along with `flex-grow` and `flex-shrink`. Like we already saw in the previous example, this property sets the initial size of the flex item - that is, width in case of row direction and height in case of column direction. Along with `flex-grow` and `flex-shrink`, this property helps decide the size of the flex item.

When you set the `flex-basis` of an item to `100px` for example, the item first occupies `100px`. And then,

1. If there's more space available AND `flex-grow` is greater than 0, the item grows to occupy more than `100px`
- OR
2. If there's less space AND `flex-shrink` is greater than 0, the item shrinks to occupy lesser than `100px`

By default, the value of `flex-basis` is `auto`, which means the size is auto-calculated based on the `width` or `height` utilities.

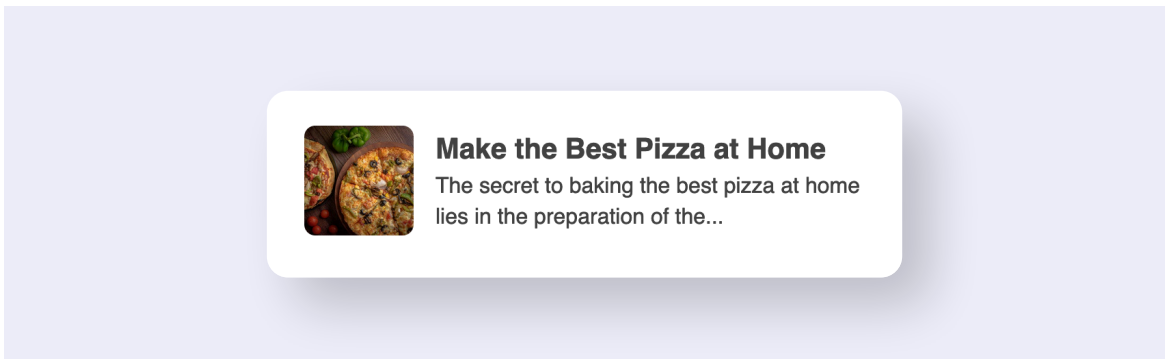
Once the `basis-*` utilities are added to the upcoming version in Tailwind CSS, you can use it similar to the `width` and `height` utilities. Some of the example values are here:



Tailwind Class	CSS Property & Value	Explanation
<code>basis-auto</code>	<code>flex-basis: auto;</code>	This is the default value. The size is auto-calculated
<code>basis-0</code>	<code>flex-basis: 0;</code>	We will soon see a use-case for 0 value
<code>basis-full</code>	<code>flex-basis: 100%;</code>	The size is 100%
<code>basis-1/2</code>	<code>flex-basis: 50%;</code>	Percentage values like <code>25%</code> , <code>50%</code> , <code>75%</code> , <code>33.33%</code> , <code>66.67%</code> and so on will be available
<code>basis-24</code>	<code>flex-basis: 6rem</code>	All the fixed values like <code>6rem</code> that are available for <code>width</code> and <code>height</code> will be available

## Blog Post Display Example 8b

Here's a blog post display example very similar to [Example 7b](#) of a large profile card. It has an image on the left and long text on the right.



## Markup

```
1 <div class="container">
2   <div>
3     <img ... >
4   </div>
5 <div> ... </div>
6 </div>
```

## Solution

```
1 <div class="container flex items-center">
2   <div class="mr-4 basis-20 flex-shrink-0">
3     <img ... >
4   </div>
5 <div> ... </div>
6 </div>
```

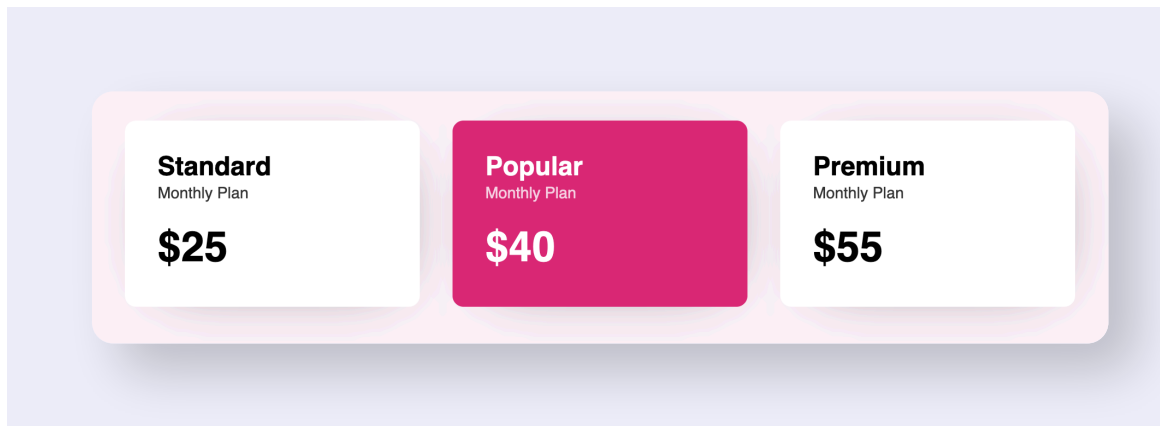
For a version lower than v3, we need the following custom styles too:

```
1 @layer utilities {
2   .basis-20 {
3     flex-basis: 5rem;
4   }
5 }
```

▶ Working Demo

## Pricing Plans Example 8c

Three equally sized blocks with margins in between is a very common pattern. With all the concepts we just learnt, this example doesn't look very hard now.



► Try it out

### Markup

```
1 <div class="container">
2   <div class="plan"> ... </div>
3   <div class="plan"> ... </div>
4   <div class="plan"> ... </div>
5 </div>
```

### Possible Solution

Set `basis-1/3` to all the flex items along with `mx-4` for spacing between the plans:

```
1 <div class="container">
2   <div class="plan mx-4 basis-1/3"> ... </div>
3   <div class="plan mx-4 basis-1/3"> ... </div>
4   <div class="plan mx-4 basis-1/3"> ... </div>
5 </div>
```

Again, for versions lower than 3.0, we need these custom styles:

```
1 @layer utilities {
2   .basis-1\3 {
3     flex-basis: 33.333333%;
4   }
5 }
```

### ▶ Working Demo

Note that though we used `basis-1/3`, the final width of each column is less than 33% of the parent because of the margins between the flex items. Each item shrinks by default to fit into the container.

Now there's a better solution to this:

### Better Solution

```
1 <div class="container">
2   <div class="plan mx-4 basis-0 flex-grow flex-shrink">... </div>
3   <div class="plan mx-4 basis-0 flex-grow flex-shrink">... </div>
4   <div class="plan mx-4 basis-0 flex-grow flex-shrink">... </div>
5 </div>
```

### ▶ Working Demo

This is better because if you add four blocks or two blocks instead of three, you don't have to change the `basis-*` value. Try removing one of the plans or adding another. They all take up equal space.

Also, in the previous solution, we don't have `flex-shrink` and `flex-grow` specified. They have their default values. It is always encouraged to set **all 3 properties** to avoid any kind of confusion. Very soon we will see how all these three utilities can be combined into just one.

## Spaces between the blocks

One thing to note is that we have added `margin` to each item to create margins in between and around the blocks. This creates some margins around the blocks too, and not just between them. The best solution hence is to use the `gap` utilities on the flex container. But the `gap` CSS property doesn't have a good browser support for flexbox yet, at the time of writing this. I encourage you to check for browser support and use it accordingly. I will talk more about this property in the Grid section of this book.

## 9 Flex Shorthand Property

Instead of using three separate utilities `flex-grow-*`, `flex-shrink-*` and `basis-*`, we can make use of a single `flex-*` shorthand utility. In the previous example, you can replace all those three CSS classes with a single class.

```
1 <div class="plan mx-4 flex-1"> ... </div>
```

Try replacing those 3 classes with just `flex-1` in the previous example and notice that everything works the same.

### Understanding Flex Concept

The `flex-*` utility classes control how flex items both grow and shrink along with specifying an initial size. In Tailwind CSS, we have four of these utility classes that cover most of the use cases.

Tailwind Class	CSS Property & Value	Explanation
<code>flex-1</code>	<code>flex: 1 1 0%;</code>	Flex item grows and shrinks as needed ignoring the initial size. If this is used on multiple items, all the items take up equal space.
<code>flex-auto</code>	<code>flex: 1 1 auto;</code>	Flex item grows and shrinks as needed considering the initial size. If this is used on multiple items, all the items take up space based on their content.
<code>flex-initial</code>	<code>flex: 0 1 auto;</code>	This is the default. The item shrinks when space is less but does not grow when there's space available. Initial size is auto-calculated.
<code>flex-none</code>	<code>flex: none;</code>	The item does not grow, nor shrink.

Along with these commonly used values available in Tailwind CSS, it's good to understand the syntax of the CSS `flex` property if you ever need to customize.

## Syntax

```
flex : <flex-grow> <flex-shrink> <flex-basis>
```

The `flex` property may be specified using one, two, or three values separated by spaces. Let's see how they are interpreted.

### One Value

The value can be

- a `<number>`: In this case, it is interpreted as `flex-grow`  
While `flex-shrink` is assumed to be `1` and `flex-basis` is assumed to be `0`  
**Example:** `flex: 1` is same as `flex: 1 1 0%`
- a `<number with units>`: It is interpreted as `flex-basis`  
While `flex-grow` is assumed to be `1` and `flex-shrink` is assumed to be `1`  
**Example:** `flex: 10rem` is same as `flex: 1 1 10rem`
- the keyword `initial`: It is interpreted as  
`flex: 0 1 auto` - the default behaviour
- the keyword `auto`: It is interpreted as  
`flex: 1 1 auto` - similar to `initial` but the item grows to occupy any additional space available
- the keyword `none`: It is interpreted as  
`flex: 0 0 auto` - neither grows nor shrinks, occupies space based on `width` and `height` properties

### Two Values

The first value must be

- a `<number>` and it is interpreted as `flex-grow`

The second value can be

- a `<number>`: It is interpreted as `flex-shrink` OR
- a `<number with units>`: It is interpreted as `flex-basis`

**Example:** `flex: 1 0` is same as `flex: 1 0 0%` AND `flex: 1 10rem` is same as `flex: 1 1 10rem`

### Three Values

The values must be in the following order:

1. a `<number>` for `flex-grow`
2. a `<number>` for `flex-shrink`
3. a `<number with units>` for `<flex-basis>`

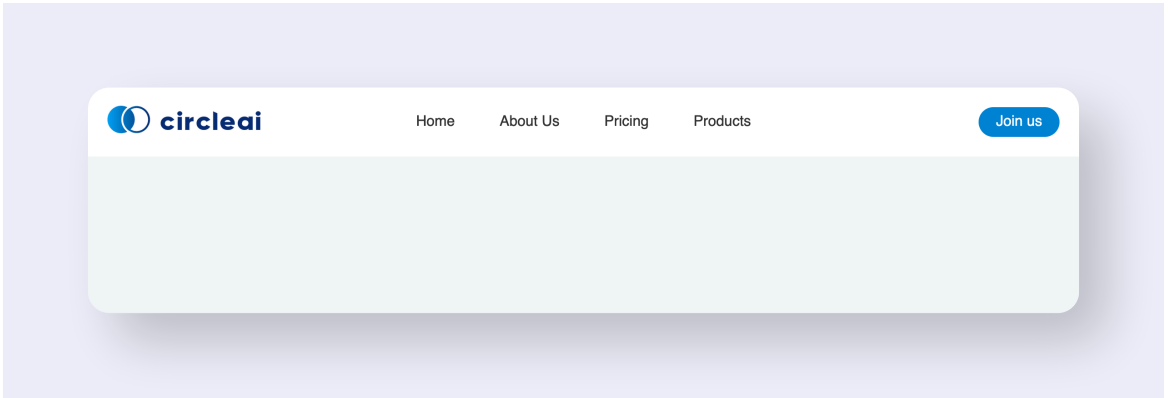
It is super hard to remember all of these at once, and the good news is - you don't have to! If you ever need to customize using the `flex` property, simply use the **three-value syntax** in the specified order to avoid confusion. If you are analysing someone else's code, use the above as a reference.

Now let's look at some examples using the `flex-*` utilities.

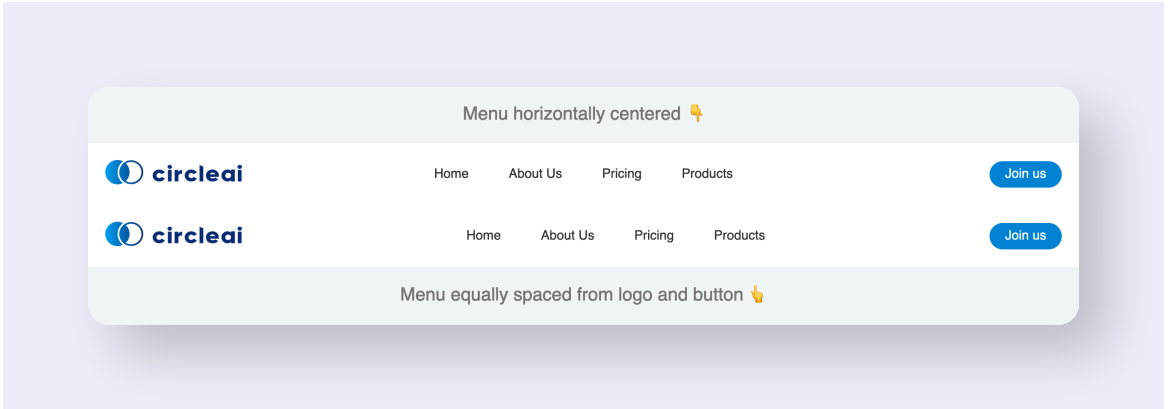
## Navigation Bar with Centered Menu Example 9a

We often come across navigation bars with a logo on the left, one or two buttons on the right and multiple menu links absolutely centered horizontally. Though it looks simple, it's not straightforward to implement.





Notice how the menu is at the exact center of the entire navbar. The distance from menu to logo and menu to button are not equal. Hence using `justify-between` is not sufficient to achieve this. Look at the difference



We want to achieve the first result. So how do we do it?

▶ Try it out

## Markup

```
1 <header>
2   <a>
3     <img ... >
4   </a>
5   <ul> ... </ul>
6   <span>
7     <button> ... </button>
8   </span>
9 </header>
```

## Solution

If the elements `a` and `span` are of same width, then `justify-between` will help us achieve the desired result. Luckily, we can *make* them occupy the same widths with what we learnt in [Example 8c](#)

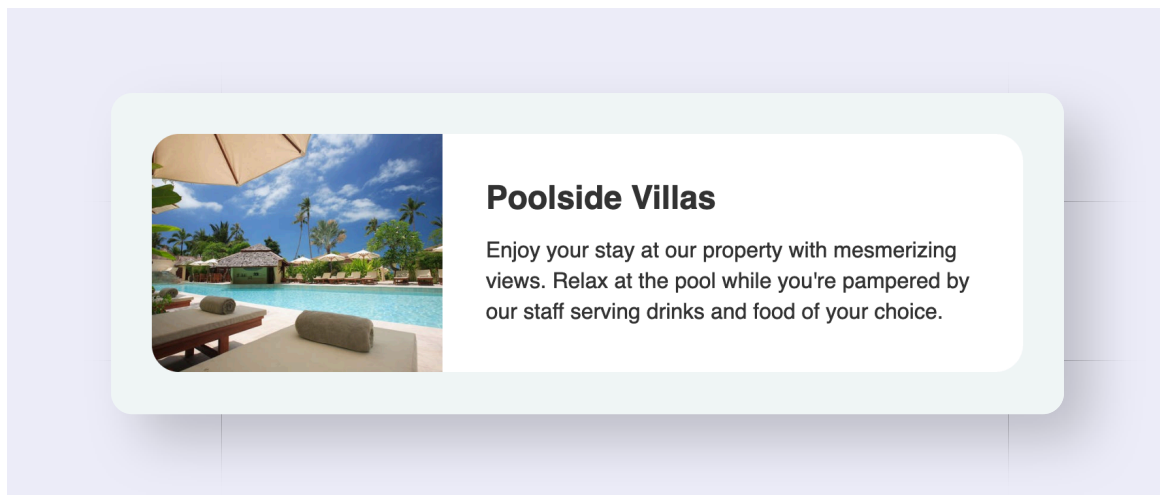
```
1 <header class="flex justify-between items-center">
2   <a class="flex-1">
3     <img ... >
4   </a>
5   <ul> ... </ul>
6   <span class="flex-1 text-right">
7     <button> ... </button>
8   </span>
9 </header>
```

Along with the `flex-1` utility, we also need `text-right` for the `span` element to push the button to the right of the `span`.

▶ Working Demo

## Image and Text in 2:1 Ratio Example 9b

You must have seen so many components with two elements placed side-by-side with widths in the ratio 2:1 or 1:2. Here's one such example. The text block is twice the width of the image and the component is flexible.



See if you can get this working using `flex-*` utility with an arbitrary value.

▶ Try it out

### Markup

```
1 <div class="container">
2   <img ... >
3   <div class="details"> ... </div>
4 </div>
```

## Solution

```
1 <div class="container flex ">
2   <img class="flex-1 w-full object-cover" ... >
3   <div class="flex-[2] details"> ... </div>
4 </div>
```

For the `img`, we have `flex-1` which is `flex: 1` or `flex: 1 1 0%`.

For the `div`, we use `flex-[2]` which translates to `flex: 2` or `flex: 2 1 0%` and hence the `div` occupies twice the width of the image.

Along with `flex-1` for the `img`, we also need `w-full` and `object-cover` to fit the image in the set space without changing the aspect ratio.

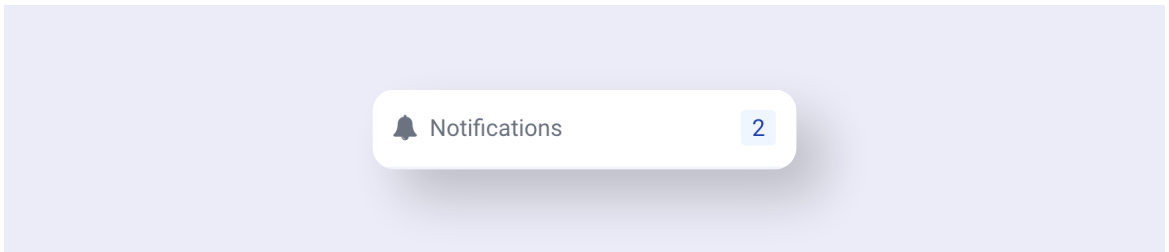
▶ Working Demo

## 10 Auto Margins

### Notifications Menu Item Example 10a

Example contributed by [Naresh](#)

Here's an example of a very small component with icon and text on left, and a count on right



#### Markup

```
1 <div class="container">
2   <i> ... </i>
3   <span class="text">Notifications</span>
4   <span class="count">2</span>
5 </div>
```

If you can wrap the icon and text within another `div`, we can achieve this look by using `justify-between`. But can you get the same look without editing this HTML?

▶ Try it out

One way to achieve this is by adding `flex-grow` to the `.text` element. Here's another solution.

## Solution

```
1 <div class="container flex align-center">
2   <i> ... </i>
3   <span>Notifications</span>
4   <span class="count ml-auto">2</span>
5 </div>
```

We have added `ml-auto` which is `margin-left: auto` to the `count` element to simply push it to the right.

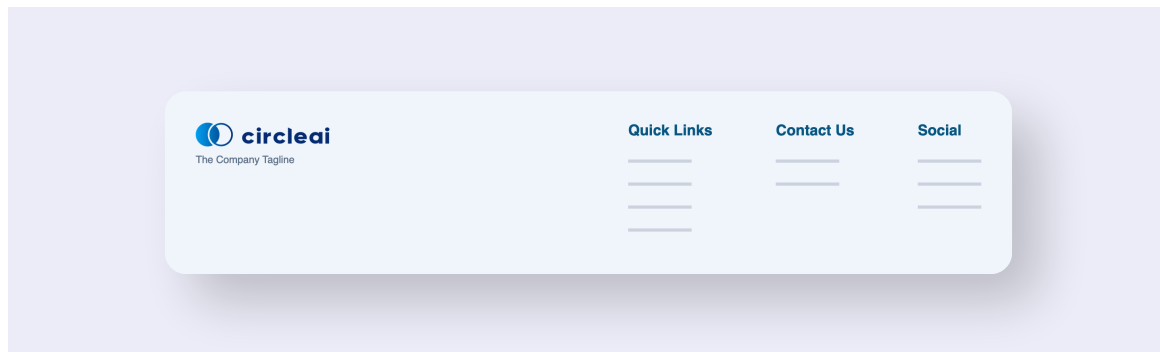
### ▶ Working Demo

The `margin` utilities can be used with flex items to extend margins to occupy the extra space. So, in the above example, the left margin occupies all the extra space on the left, pushing the element to the right.

If you can recall, we used `m-auto` to center a single flex item within its container in [Example 4e](#) - Solution 2. This works the same way.

## Footer with Multiple Columns Example 10b

This is another common footer structure with logo on the left and a few columns "pushed" to the right.



Based on what you saw in the previous example, can you get this result using auto margins?

▶ Try it out

## Markup

```
1 <footer>
2   <div class="footer-col"> ... </div>
3   <div class="footer-col"> ... </div>
4   <div class="footer-col"> ... </div>
5   <div class="footer-col"> ... </div>
6 </footer>
```

## Solution

We need to "push" the 2nd column to right

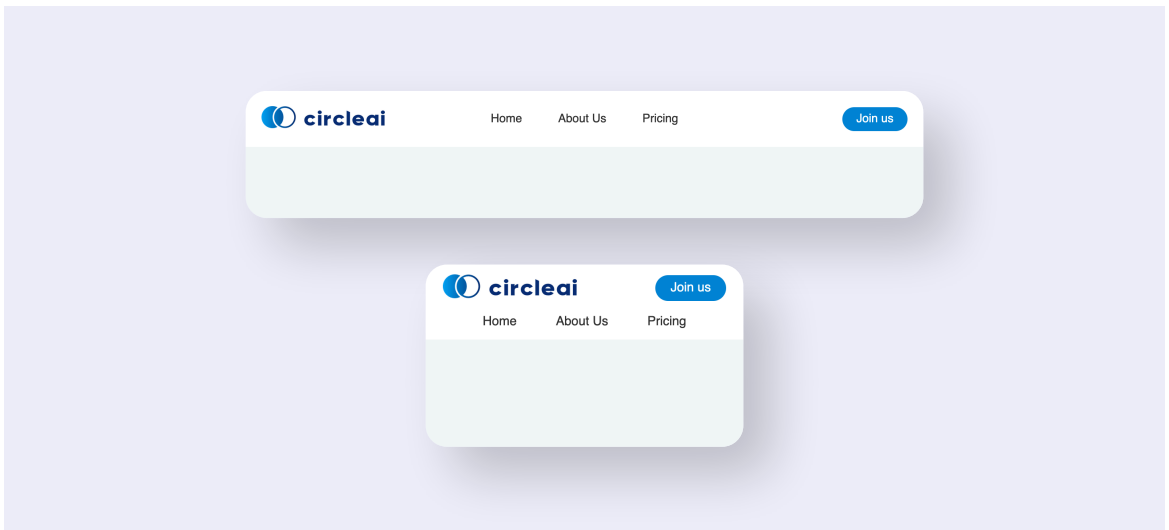
```
1 <footer class="flex">
2   <div class="footer-col"> ... </div>
3   <div class="footer-col ml-auto"> ... </div>
4   <div class="footer-col"> ... </div>
5   <div class="footer-col"> ... </div>
6 </footer>
```

▶ Working Demo

# 11 Order

## Responsive Navigation Bar Example 11a

Let's look at our [Example 9a](#) once again and make it responsive now. Assume that you have just 3 links in the navigation bar and you want those links to appear in the second row on mobile screens.



### Markup

```
1 <header>
2   <a>
3     <img ... >
4   </a>
5   <ul> ... </ul>
6   <span>
7     <button> ... </button>
8   </span>
9 </header>
```



Since we follow the mobile first approach in Tailwind CSS, on small screens we first need to do two things:

1. Change the order of the `ul` element to appear last
2. Make the `ul` element occupy full width

Then using the `md:` prefix,

1. Set the order of `ul` back to 0
2. Set the width of `ul` back to auto

If you are aware of the `order-*` utilities, give this a shot.

▶ Try it out

### Solution

```
1 <ul class="order-last flex-[100%] md:order-none md:flex-auto"> ...
  </ul>
```

▶ Working Demo

The only new utility here is the `order` utility.

## Understanding Order Concept

The `order` property is also used on a **flex item**. The value can be any number - positive or negative. The items with greater `order` value appear later on the web page compared to the items with lesser value irrespective of their appearance in the markup.

If no `order` is specified, by default the value is `0` for all the elements and they follow the same order as they appear in HTML.

Some of the common utilities for `order` are here:

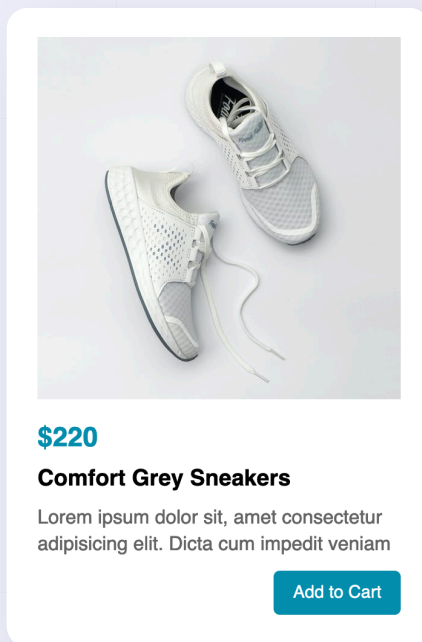
Tailwind Class	CSS Property & Value	Explanation
<code>order-1</code>	<code>order: 1;</code>	Any number from 1 to 12 are available similarly using <code>order-2</code> , <code>order-3</code>
<code>order-first</code>	<code>order: -9999;</code>	The item gets placed at the beginning because the value is a large negative number.
<code>order-last</code>	<code>order: 9999;</code>	The item gets placed at the end because the value is a large positive number.
<code>order-none</code>	<code>order: 0;</code>	This is the default.

Hence when we set `order-last` to the `u1` element in the example above, only that element is removed from the normal flow and placed at the end. And for large screens, we change it back to normal flow using `order-none`.

## 12 Align Self

### Product Display Example 12a

This is a card component using `flex-col`. By default, all the elements are stretched full width (along the cross axis). But you want the button alone to be pushed to the right instead of stretching full width.



► Try it out

## Solution

```
1 <div class="container flex flex-col">
2   <img ... >
3   <span> ... </span>
4   <h3> ... </h3>
5   <p> ... </p>
6   <button class="self-end"> ... </button>
7 </div>
```

### ▶ Working Demo

We are using the utility class `self-end`, which translates to `align-self: flex-end`. This makes only the `button` align to the end along the cross axis. Remember, [main axis and cross axis?](#)

## Understanding Align Self Concept

The `self-*` utilities for a flex item are similar to the `items-*` utilities. These classes override the `items-*` classes applied to the parent container. Note that:

1. `self-*` classes are applied to a flex item, whereas `items-*` are applied to a flex container
2. `self-*` takes effect only on the item it's applied to, whereas `items-*` works for all the flex items within the container.

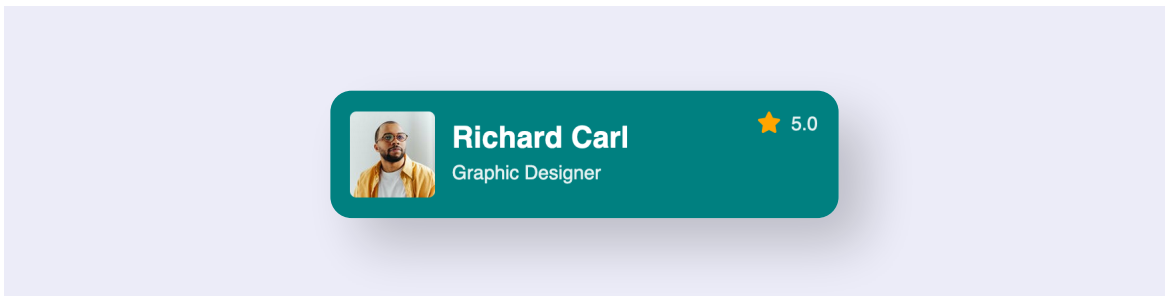
The available utility classes are also similar to `items-*`.

Tailwind Class	CSS Property & Value	Explanation
<code>self-stretch</code>	<code>align-self: stretch;</code>	The item is stretched to fill the container along the cross axis
<code>self-center</code>	<code>align-self: center;</code>	The item is placed at the center of the container along the cross axis
<code>self-start</code>	<code>align-self: flex-start;</code>	The item is placed at the beginning of the container ( <i>at the top for <code>row</code> direction and at the left for <code>column</code> direction</i> )
<code>self-end</code>	<code>align-self: flex-end;</code>	The item is placed at the end of the container ( <i>at the bottom for <code>row</code> direction and at the right for <code>column</code> direction</i> )
<code>self-baseline</code>	<code>align-self: baseline;</code>	The item is positioned such that the base aligns to the end of the container ( <i>applies only for <code>row</code> direction</i> )

Let's look at another use-case.

## Profile with Rating Example 12b

This is a small variation to the profile card we saw in [Example 4b](#). This one has a rating at the top right corner of the card. While the image and text are center aligned vertically, the rating is aligned to the top.



Can you get this working?

▶ Try it out

## Markup

```
1 <div class="container">
2   <img ... >
3   <div> ... </div>
4   <div class="rating"> ... </div>
5 </div>
```

## Solution

You need two Tailwind classes for the `rating` div - one for pushing it to the right (*alignment along main axis*) and another for aligning it at the top (*alignment along cross axis*).

```
1 <div class="container flex items-center">
2   <img ... >
3   <div> ... </div>
4   <div class="rating ml-auto self-start"> ... </div>
5 </div>
```

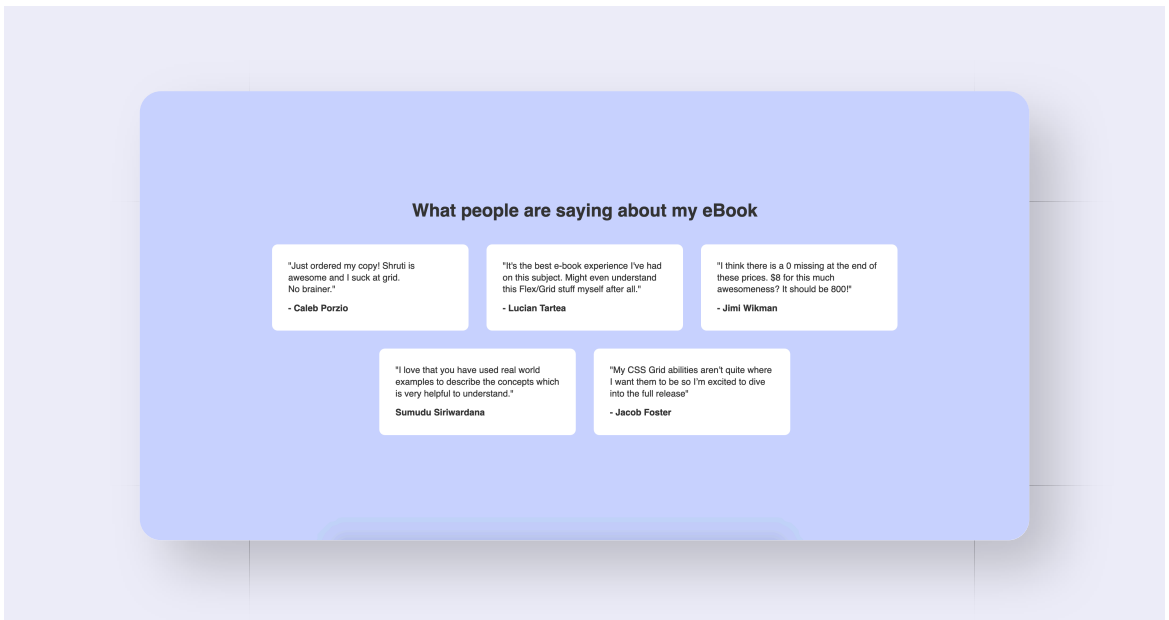
▶ Working Demo

This example might help you understand why we have a CSS property `align-self` (*self-\* utility classes*) but no property like `justify-self` because we can simply use auto margins to space out or align a single item along the main axis.

## 13 Align Content

### Full Page Testimonials Section Example 13a

Let's say you have a few testimonial cards as flex items wrapped in multiple rows. You want these items to be center aligned vertically in a full height page.



You might think this is what `items-center` does. But no. That works only for single row flex items.

► Try it out

## Markup

```
1 <div class="container">
2   <div class="testimonial">... </div>
3   <!-- Four more testimonial divs -->
4 </div>
```

One of the options is to wrap all the `.testimonial` elements in another `div` and center align that `div` vertically. But that's an unnecessary addition of an element to the DOM.

## Solution

```
1 <div class="container flex flex-wrap justify-center content-center">
2   ...
3 </div>
```

### ▶ Working Demo

All we need to do is use the `content-center` utility class instead of `items-center`.

## Understanding Align Content Concept

The `content-*` utilities are used on the **flex container** for aligning **multi-line** flex items along the cross axis. It works only for flex items that flow into multiple rows or columns. The available utilities are mentioned below:



Tailwind Class	CSS Property & Value	Explanation
<code>content-start</code>	<code>align-content: flex-start;</code>	The items are packed at the beginning of the container
<code>content-end</code>	<code>align-content: flex-end;</code>	The items are packed at the end of the container
<code>content-center</code>	<code>align-content: center;</code>	The items are packed at the center of the container
<code>content-between</code>	<code>align-content: space-between;</code>	The rows / columns are spaced out as much as possible with first line at the beginning and last line at the end
<code>content-around</code>	<code>align-content: space-around;</code>	Space at the beginning and the end are half as much as space between the lines
<code>content-evenly</code>	<code>align-content: space-evenly;</code>	Space at the beginning, end and between the lines are same

*Note: This property is very rarely used. So it's totally okay if you cannot remember it 😊*

## 14 Inline Flex

### Social Media Icons Example 14a

Example inspired by [Ahmad Shadeed's Article](#)

Let's say you want a row of rounded icons with each icon placed at the exact center within the circle like this.



To center each icon within the circle, we can add `flex` to the circle and center using `justify-center` and `items-center`. But that leaves us with the circles stacked one below the other, instead of next to each other

▶ Try it out

### Markup

```
1 <a class="icon flex justify-center items-center" href="#">
2   <i class="fa fa-twitter"></i>
3 </a>
4 <a class="icon flex justify-center items-center" href="#">
5   <i class="fa fa-linkedin"></i>
6 </a>
7 <a class="icon flex justify-center items-center" href="#">
8   <i class="fa fa-github"></i>
9 </a>
```

## Solution

Now simply change `flex` to `inline-flex`

```
1 <a class="icon inline-flex justify-center items-center" href="#">
2   ...
3 </a>
4 <a class="icon inline-flex justify-center items-center" href="#">
5   ...
6 </a>
7 <a class="icon inline-flex justify-center items-center" href="#">
8   ...
9 </a>
```

▶ Working Demo

## Understanding Inline Flex Concept

All the utilities that we have seen so far, either applied on flex container or on flex items, affect the **flex items** in one or other way - by changing their dimensions, position or alignment within the container. But the utility `inline-flex` does not affect the flex items. It makes the **flex container** itself behave like an `inline` element instead of a `block` element.

Tailwind Class	CSS Property & Value	Explanation
<code>inline-flex</code>	<code>display: inline-flex;</code>	Makes the flex container itself behave like an inline element

That's how we make the icons appear next to each other (inline) in our previous example where each icon itself is a flex container.

## Flexbox Unlocked!

Reaching till this point of the book means you have learned about all the things you can do with flexbox's utilities of Tailwind CSS. Yay! 💪

You might not remember everything and that's perfectly normal. Since you learned through real world examples, you need just a few more revisions before you start using flexbox like a pro. And you can always look back at these examples for reference.

Let's look at some comprehensive examples each involving multiple flexbox containers in each component.

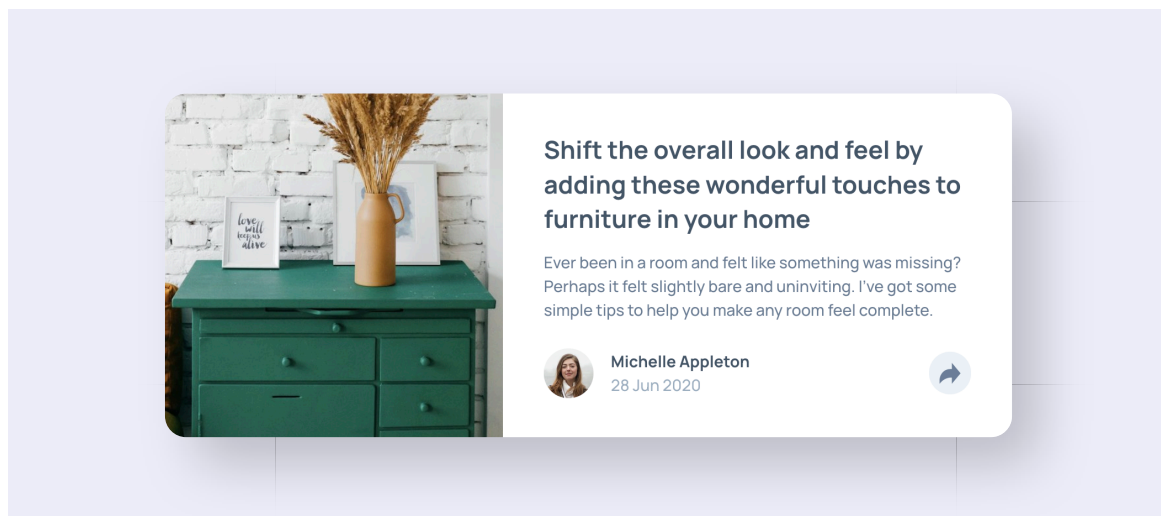
# Comprehensive Examples for Flexbox

## Article Preview

Example 15a

Challenge from [Frontend Mentor](#)

This article preview component is a challenge from the Frontend Mentor website.



This is an example of using flexbox within flexbox. The whole card is a flex container with with image occupying 40% width and text block occupying 60%. Then the footer in the text block is another flex container with the author image, name & date and the share icon being the flex items.

Following the mobile-first approach, make this component responsive, by setting the outer flex container's direction to `column` and for larger screens, change it to `row`.

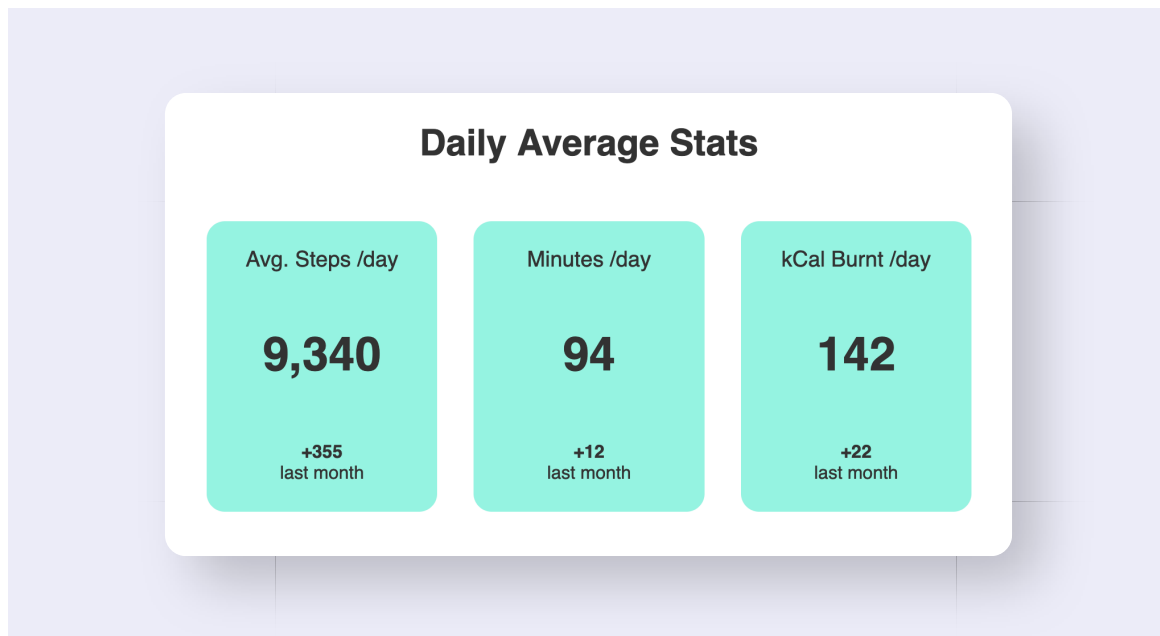
▶ Try it out

**HINT:** Use the utilities `flex`, `flex-col`, `align-items`, `ml-auto` and arbitrary values for `flex-*` to achieve the 40% and 60% widths.

▶ Working Demo

## Fitness Report Example 15b

Here's a simple fitness report component to test your newly gained flexbox skills



You again need to use flexbox within flexbox, but this time, column direction within row direction. Bonus points if you can make it responsive *without* using media queries.

[▶ Try it out](#)

**HINT:** Use the utilities `flex`, `flex-wrap`, `flex-*` for the outer flexbox. Use `flex-col`, `justify-*` and `min-w-*` to allow the blocks to wrap.

[▶ Working Demo](#)

## Tweet Example 15c

There's so much to learn from a single "tweet" design. This is an exact mockup of a tweet in the timeline on the Twitter web app (except for the font).



Here you will need to create three flexbox containers! One for the entire tweet. Two for the name, handle, date and options. And third one for the row with actions having "reply", "retweet" etc.

[▶ Try it out](#)

**HINT:** Use the utilities `flex`, `justify-*`, `items-*` and auto margins.

[▶ Working Demo](#)

Whoa! This really completes almost everything you can do with CSS flexbox. Take some time to digest all this, practice some more with other components and layouts you observe. And then come back to learn CSS Grid.

**Caution:** Grid is slightly more complex than flexbox! Be prepared.



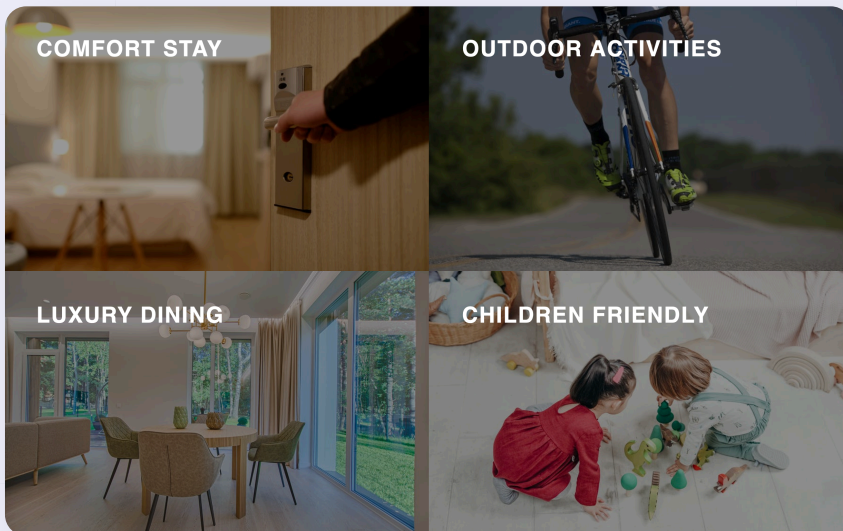
# Grid

## 16 Display Grid & Grid Template Columns

Let's start off with CSS Grid looking at the simplest possible example.

### Full Page Gallery Example 16a

Let's say you want to create a gallery page for a resort, listing all the albums in a **grid** fashion occupying full screen like this.



This is surely possible using `float`, `table` or `flex`. But if you want a simpler solution, grid is the best option. If you already have an idea about grid or if you wish it to try this layout any other way, feel free it to give it a shot.

► Try it out

## Markup

```
1 <div class="container min-h-screen">
2   <div class="item"> ... </div>
3   <!-- Three more items -->
4 </div>
```

## Solution

Now you need to add two utility classes to the `.container` element to arrange the child elements in a grid form.

```
1 <div class="container min-h-screen grid grid-cols-2">
2   ...
3 </div>
```

### ▶ Working Demo

We just added `grid` and `grid-cols-2` and do you see what happened? Each item occupies 50% width and all the items add up to fill the entire vertical space (`min-height` of the container is `100vh`). Now let's understand these utilities one by one.

## Understanding Display Grid Concept

While **flexbox** helps us arrange elements in one dimension (row or column), **grid** is a method that helps us arrange and align elements in both the dimensions with rows *and* columns. Similar to flexbox, we can control the the size, alignment, placement and order of these elements using grid. Here again, we need at least two elements - a parent element called **grid container** and at least one child element called **grid item**.

In our above example, adding `grid` class makes the `.container` element a grid container.

Tailwind Class	CSS Property & Value	Explanation
<code>grid</code>	<code>display: grid;</code>	Setting the <code>display</code> property of an element to <code>grid</code> makes it a grid container

But this utility alone does not make any difference because it creates one column by default. We need the next property `grid-cols-*` to specify the number of columns.

## Understanding Grid Template Columns Concept

The utilities `grid-cols-*` is used to specify how many columns you need and of what size each. Majority of the use cases for grid require creating equal width columns, and hence Tailwind provides these utility classes where you can create one to twelve columns of equal width.

Tailwind Class	Explanation
<code>grid-cols-1</code>	Creates one grid column occupying full width of the container
<code>grid-cols-2</code>	Creates two grid columns occupying 50% width each
<code>grid-cols-3</code>	Creates three grid columns occupying 33.33% width each

We have such classes available from `grid-cols-1` until `grid-cols-12`. But sometimes we need grid columns with unequal widths. To understand how to create them using Tailwind, we need to first understand how this works in CSS.

## The CSS Property `grid-template-columns` & Values

**CSS Syntax:**

```
grid-template-columns: <value> <value> ...
```

Using the CSS property `grid-template-columns`, we can specify the widths of each column in `%`, `px`, `rem` etc., separated by spaces. The number of individual values you specify will be the number of columns created. The previous example of full page gallery can be achieved in CSS using:

```
grid-template-columns: 50% 50%
```

This creates two columns of 50% width each and the rows are automatically created. But if we need something like 40% and 60% width columns, in we can say:

```
grid-template-columns: 40% 60%
```

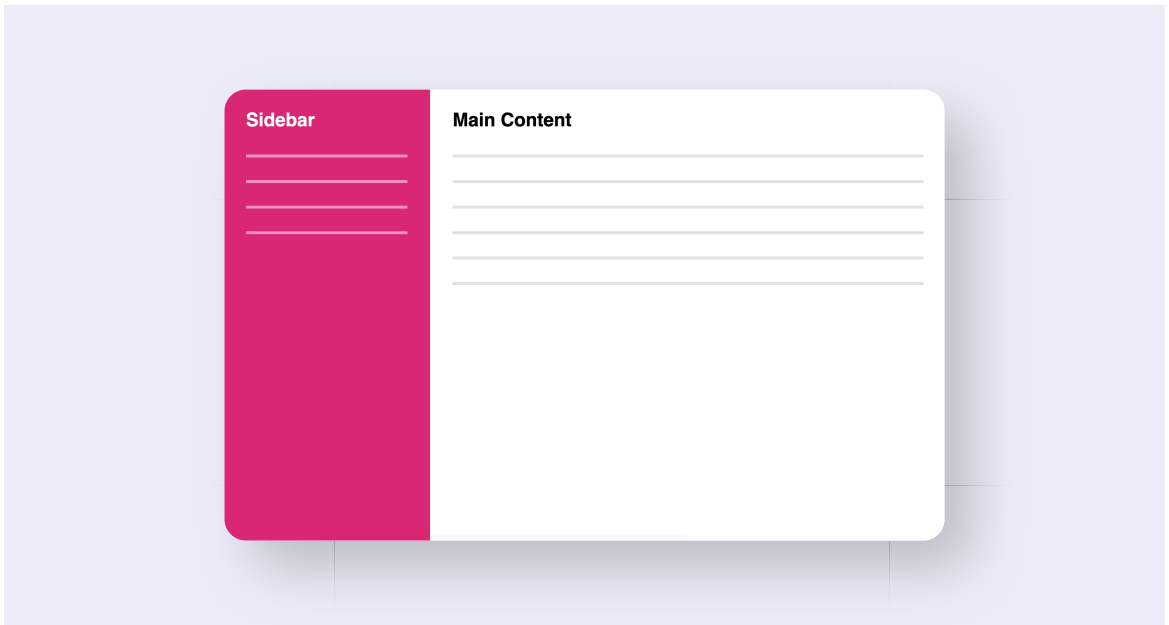
Since we don't have a Tailwind utility for such values, we can use arbitrary values for achieving the same output:

```
grid-cols-[40%,60%]
```

Note that *space* is replaced by *comma* in square brackets. The syntax of `grid-template-columns` gets more complex for complex layouts. Let's look into each of them with appropriate examples.

## Layout with Sidebar Example 16b

This is a common layout with a sidebar on left and main content on the right. There are multiple ways of achieving this, but **grid** makes it simplest.



## Markup

```
1 <div class="container min-h-screen">
2   <div class="sidebar"> ... </div>
3   <div class="main"> ... </div>
4 </div>
```

## Solution

We need two columns here - `.sidebar` with a fixed width and `.main` that takes up the remaining space. This is possible with an `fr` unit in `grid-template-columns`. Here's the Tailwind solution:

```
1 <div class="container min-h-screen grid grid-cols-[22rem,1fr]">
2   ...
3 </div>
```

Along with `grid` class, we added `grid-cols-[22rem, 1fr]` which translates to `grid-template-columns: 22rem 1fr`. So we get two columns - first one with fixed `22rem` width and second one which occupies the remaining space.

▶ Working Demo

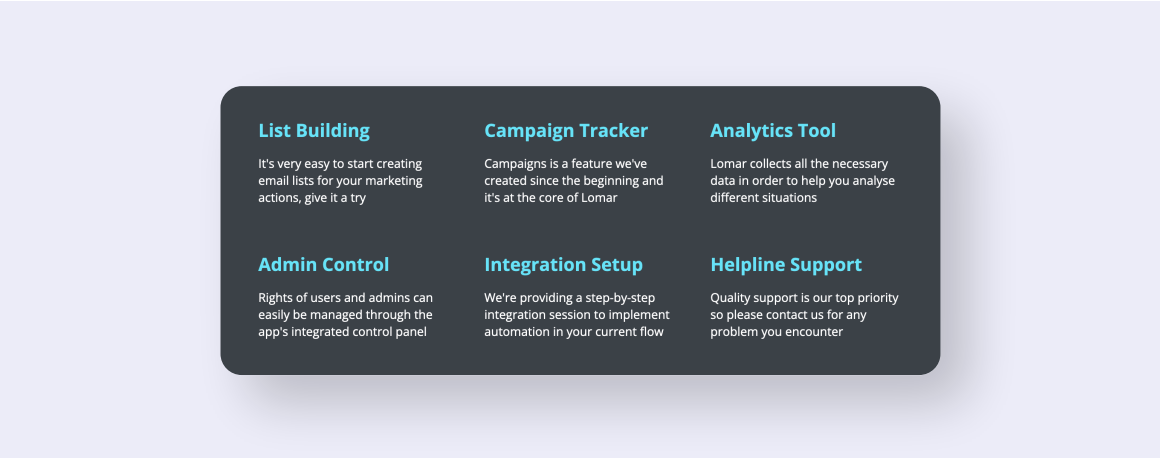
## The `fr` Unit

The `fr` is short for **fraction** representing fraction of the remaining space. In flexbox, we can set `flex-grow` of items to a value greater than 0 to make those elements occupy fractions of the remaining space in the parent container right? The `fr` unit is quite similar to that. In our above example, there's only one column with the `fr` unit, so that column takes up 100% of the remaining space. We will soon see most of our examples involving `fr` units.

## Services Grid Example 16c

Example inspired by [Inovatik Template](#)

This is a classic example of grid - listing services or features in a grid format with equal width columns.



Can you try this out?

▶ Try it out

### Solution

We need three columns of equal width. In Tailwind, we have already seen how simple it is to do the same.

```
1 <div class="container grid grid-cols-3">
2   <div class="item"> ... </div>
3   <!-- Five more items here -->
4 </div>
```

▶ Working Demo

### CSS Solution

But if we have to achieve the same in CSS, we can use the `1fr` unit. The `1fr` unit helps us distribute the remaining space (which is all the space in this example) proportionately.

```
1 .container {
2   display: grid;
3   grid-template-columns: 1fr 1fr 1fr;
4 }
```

So, you can repeat the `1fr` unit as many times as the number of equal sized columns you need. And, to avoid repetition, we can use the `repeat()` function in CSS. This function takes in two inputs. The first one is the number of times you want to repeat and second one being the value you want to repeat.



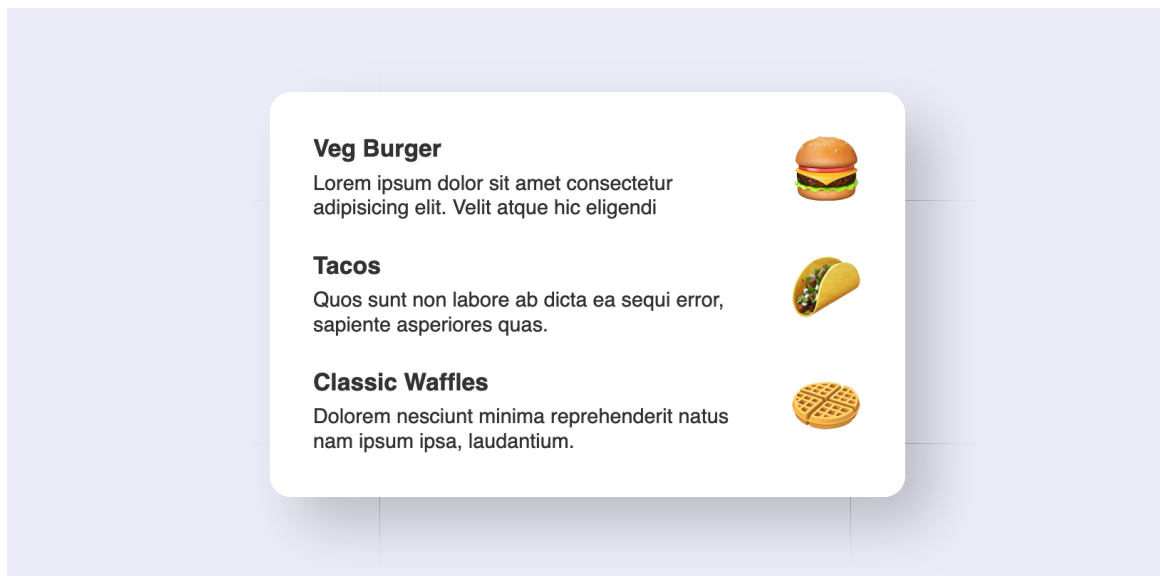
## Better Solution

```
1 .container {  
2   display: grid;  
3   grid-template-columns: repeat(3, 1fr);  
4 }
```

It's good to understand how the `grid-cols-*` Tailwind utilities work under the hood. But we're not fully there yet.

## Quick Bites Menu Example 16d

Usually restaurant menus are displayed in a grid fashion. Here's one such example with the item name and description on the left and a picture on the right. Here, we want the items in first column to occupy as much space as possible and the pictures to occupy only as much space as needed.



► Try it out

### Markup

```
1 <div class="container">
2   <div class="item"> ... </div>
3   <span class="icon"> ... </span>
4   <div class="item"> ... </div>
5   <span class="icon"> ... </span>
6   <div class="item"> ... </div>
7   <span class="icon"> ... </span>
8 </div>
```

## Solution

We need two columns of unequal width, so we can use `grid-cols-*` with arbitrary values to specify two values separated by commas. As you might have guessed, the first value is `1fr`. You can specify a fixed width for the second column. But what's even better is to use the keyword `auto`. This keyword lets the content of items decide the size of the column / row.

```
1 <div class="container grid grid-cols-[1fr,auto]">
2   ...
3 </div>
```

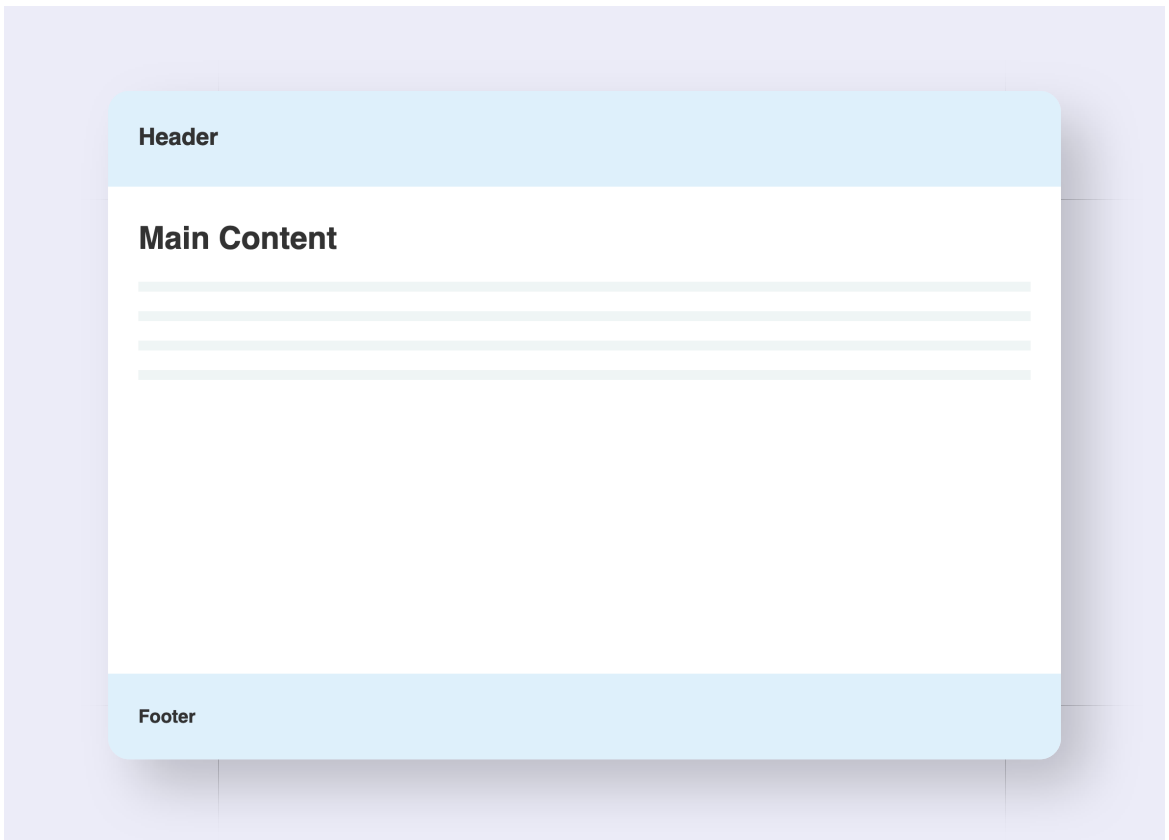
### ▶ Working Demo

So far we used fixed units, percentage values, `fr` units and the `auto` keyword for specifying `grid-cols-*` arbitrary values apart from the Tailwind utilities available. There are a few more options that we'll cover under the [Advanced Grid Template Values](#) topic.

## 17 Grid Template Rows

### Sticky Footer with Grid Example 17a

We already saw how to make a [sticky footer using flex](#). Here's a similar example with an additional header element. If you can recall, we used `flex-col` and `flex-grow` to create this.



Try this out with grid if you can guess how it's done.

▶ Try it out

## Markup

```
1 <div class="container min-h-screen">
2   <header> ... </header>
3   <div class="main"> ... </div>
4   <footer> ... </footer>
5 </div>
```

## Solution

Look at the example and observe that we have a single column but multiple rows. This can be specified using `grid-rows-*` instead of `grid-cols-*`. We need 3 rows - first and third rows with `auto` height and the second row occupying all the remaining height.

```
1 <div class="container min-h-screen grid grid-rows-[auto,1fr,auto]">
2   ...
3 </div>
```

### ▶ Working Demo

**Note:** This demo works as long as you have only three elements `header`, `footer` and `.main`. If you add an additional element at the same level, it breaks. Whatever additional content you want, you need to add it within the `.main` div.

## Understanding Grid Template Rows Concept

The property `grid-template-rows` in CSS is used to specify how many rows you need and of what size each. Similar to `grid-template-columns`, the height of rows can be specified in `%`, `px`, `rem` or any valid value for height separated by spaces. The number of individual values you specify will be the number of rows created.

In all the previous examples we got multiple rows even without using this property. That's how grid works. Rows are automatically created to accommodate the additional items. This is called **implicit grid**. Similarly in our above example, one column is automatically created that occupies full width by default. We'll look into implicit grids and their sizing in a while.

In the above example we needed rows of different heights, hence we used arbitrary values. But if we need equal height rows, we have `grid-rows-*` utility classes for that in Tailwind.

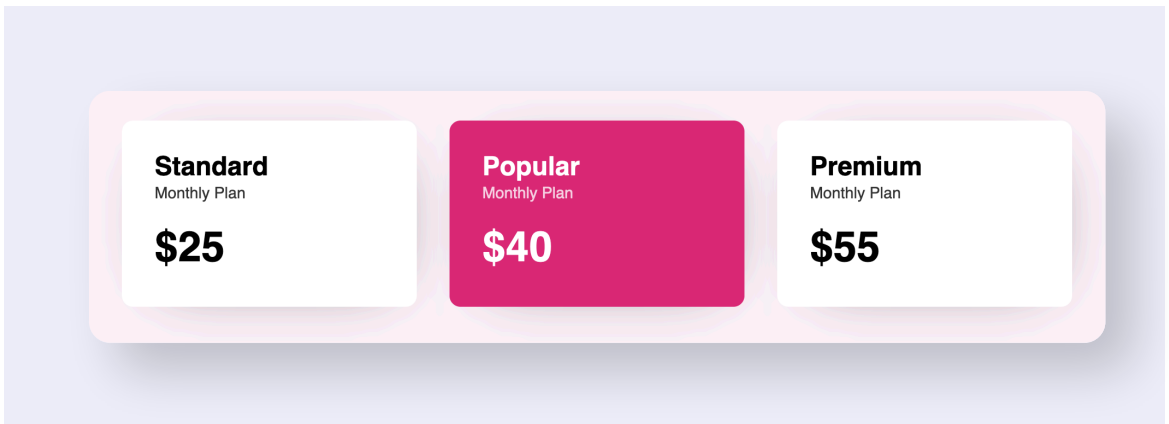
Tailwind Class	Explanation
<code>grid-rows-1</code>	Creates one grid row occupying full height of the container
<code>grid-rows-2</code>	Creates two grid rows occupying 50% height each
<code>grid-rows-3</code>	Creates three grid rows occupying 33.33% height each

We will also look at more examples using `grid-rows-*` after covering a few more concepts. For now, I'm sure you have a basic idea. And of course, you can use both `grid-col-*` and `grid-rows-*` at the same time.

## 18 Gap

### Pricing Plans with Grid Example 18a

Let's look at the same pricing plans example once again, this time using grid. By now you know how to create three equal sized blocks (or equal sized columns) with grid. But let's also look at how to add those spaces between those blocks.



First try creating this with grid, and see how you can add some spaces between the items.

▶ Try it out

#### Markup

```
1 <div class="container">
2   <div class="plan"> ... </div>
3   <div class="plan"> ... </div>
4   <div class="plan"> ... </div>
5 </div>
```

## Solution

```
1 <div class="container grid grid-cols-3 gap-x-8">
2   ...
3 </div>
```

We have added a `gap-x-8` class to add a spacing of `2rem` between the columns.

▶ Working Demo

## Understanding Column Gap Concept

The `gap-x-*` utilities set the size of the horizontal gap (also known as gutters) between columns. Like I mentioned earlier in this book, this property can be used with flexbox too, but doesn't have good browser support yet. With grid however, it is better supported. Some of the common utilities for `gap-x-*` are here:

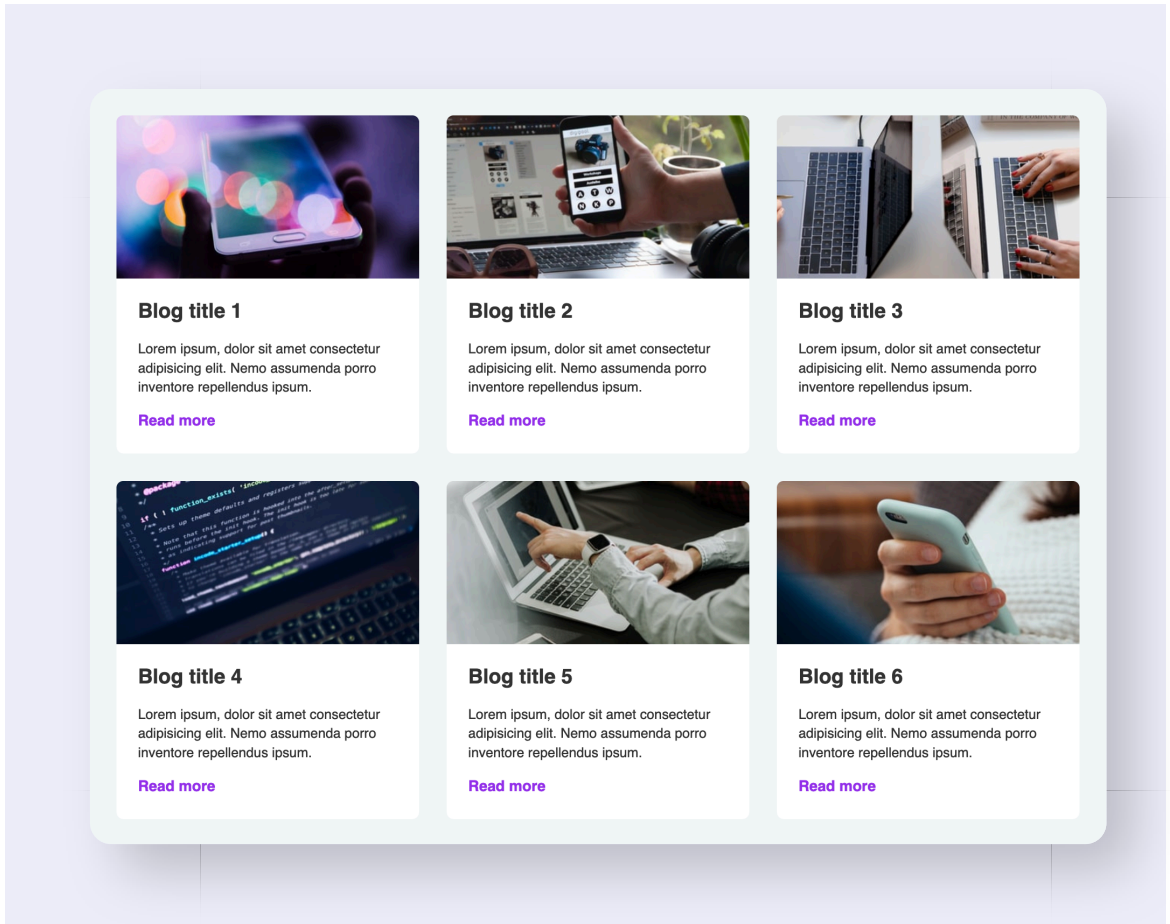
Tailwind Class	CSS Property & Value	Explanation
<code>gap-x-0</code>	<code>column-gap: 0;</code>	Gap between columns is <code>0</code>
<code>gap-x-4</code>	<code>column-gap: 1rem;</code>	Gap between columns is <code>1rem</code>
<code>gap-x-6</code>	<code>column-gap: 1.5rem;</code>	Gap between columns is <code>1.5rem</code>
<code>gap-x-8</code>	<code>column-gap: 2rem;</code>	Gap between columns is <code>2rem</code>

For all the available `gap-x-*` utilities, [check the docs](#).



## Blog Posts Display Example 18b

This is a classic use case for grid - display of blog post cards in a grid format with horizontal and vertical spacing between each card. Let's create this layout using grid and also make it responsive.



Check the below link to see how this layout is created with Grid and made responsive using the mobile first approach. Now see if you can add some horizontal and vertical spacing between the items using `gap-x-*` and a similar property `gap-y-*`.

[▶ Try it out](#)

## Markup

```
1 <div class="container">
2   <div class="item"> ... </div>
3   <!-- Five more item cards -->
4 </div>
```

## Responsive Solution

Using the mobile first approach, we just add a `grid` class at first. This automatically creates one column. At `sm` breakpoint, we change that to two columns using `grid-cols-2` and at `md` breakpoint, we change it to three columns using `grid-cols-3`. We also add spacing of `2rem` between columns and rows with `gap-x-8` and `gap-y-8`.

```
1 <div class="container grid sm:grid-cols-2 md:grid-cols-3 gap-x-8 gap-
  y-8">
2   ...
3 </div>
```

### ▶ Working Demo

These gap utilities make it very easy to create spacing only between the items and not around them. If we had to use `margin` utilities to do the same, we would have to change those margins at every breakpoint too. Or we would have to use negative margins on the container. We can also combine both the column and row gap utilities into one single `gap-*` utility.

## Better Solution

```
1 <div class="container grid sm:grid-cols-2 md:grid-cols-3 gap-8">
2   ...
3 </div>
```

## Understanding Row Gap Concept

The `gap-y-*` utilities set the size of the vertical gap (also known as gutters) between rows. Again, this property can be used with flexbox too, but doesn't have good browser support yet. With grid however, it is better supported. The available utilities are similar to that of `gap-x-*`. For all the available values, [check the docs](#).

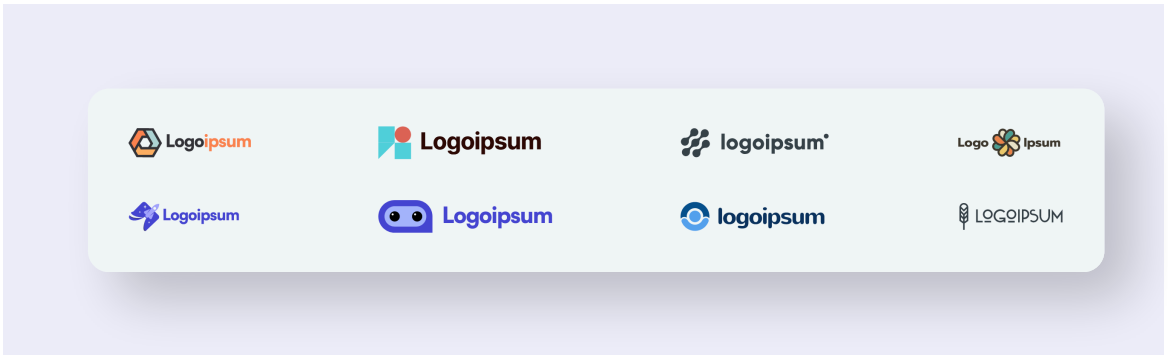
## Understanding Gap Concept

The utility `gap` is used to set the same spacing between rows and columns at once. The available utilities are similar to that of `gap-x-*` and `gap-y-*`. For all the available values, [check the docs](#).

## 19 Justify Content

### Featured Logos in a Grid Example 19a

We have seen a similar [example of logos with flexbox](#). But with flex, you can only align logos in one direction. You cannot control the alignment in the other direction. That is, we can space them out horizontally and align them in each row, but we cannot align them across multiple rows using flexbox. So, if you want them to be displayed in a neat grid format, you need to use grid. For this example, we also want the logos to be spaced out to occupy the full width of the container.



If we use `grid-cols-4` for this, the container will be divided in four equal columns, so that will not help us space out the logos to the extreme ends. Instead, let's use `grid-cols-[auto,auto,auto,auto]`, which is same as `grid-cols-[repeat(4,auto)]`. Next, we need to space them out fully.

► Try it out

In the above link, notice how there's a little gap on the right. We need to change this to make the logos stretch end-to-end.

## Markup

```
1 <div class="container grid grid-cols-[repeat(4,auto)] gap-12">
2   <img ... >
3   <!-- Seven more img elements -->
4 </div>
```

## Solution

We need one more utility class now - `justify-between` to space out the columns. This is similar to what we did with flexbox.

```
1 <div class="container grid grid-cols-[repeat(4,auto)] gap-12 justify-
  between">
2   ...
3 </div>
```

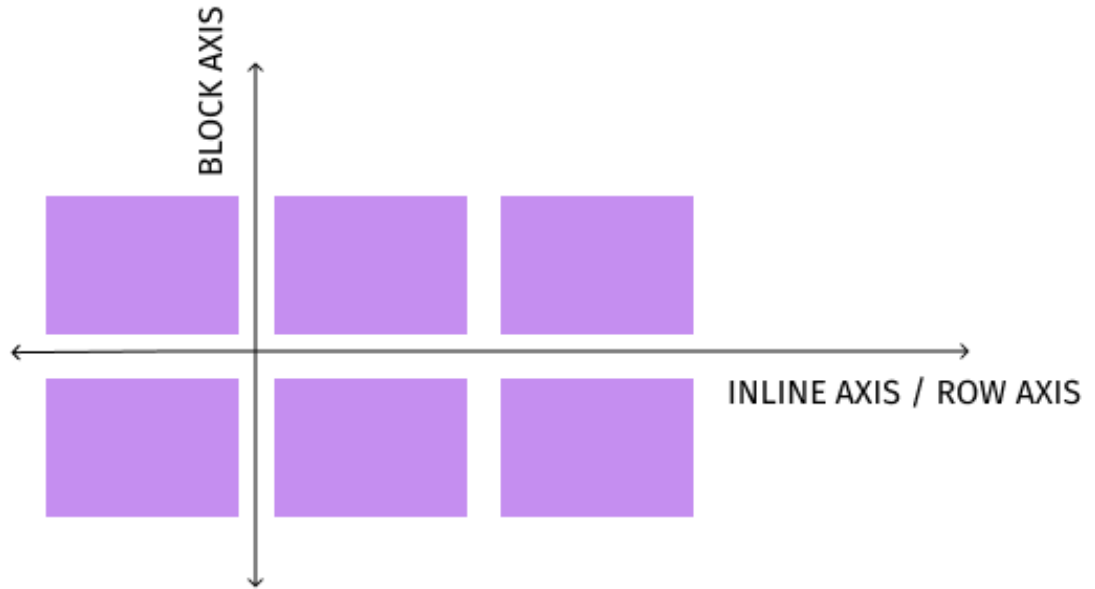
### ▶ Working Demo

This is not yet responsive. We can do that without adding media queries. We will see that in a while. And now, to notice the difference between flex and grid, change the `grid` and `grid-cols-*` utilities to `flex` and `flex-wrap` instead.

## Understanding Justify Content in Grid Concept

Before we talk about this property with reference to Grid, you need to know two more terms. In flexbox, you have the **main axis** and the **cross axis**. Similarly, while working with grid, you have the **inline axis** and the **block axis**.

Inline axis is the direction in which *inline* elements like `span` and `img` get placed. So usually it's the row direction. While block axis is the direction in which *block* elements like `div` and `section` get placed. So it's the vertical axis.



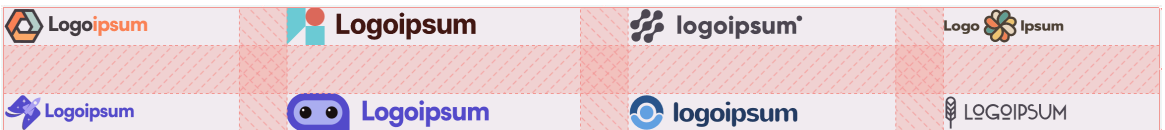
The `justify-*` utilities are used to control placement of the grid items along the *inline/row axis* - which is the horizontal direction. In simple terms, this is used to control the placement and spacing between the grid columns. `justify-between` is one of the available utilities we just used. Some more utilities are mentioned below:

Tailwind Class	CSS Property & Value	Explanation
<code>justify-start</code>	<code>justify-content: flex-start;</code>	All columns are placed at the beginning of the container
<code>justify-end</code>	<code>justify-content: flex-end;</code>	All columns are placed at the end of the container
<code>justify-center</code>	<code>justify-content: center;</code>	All columns are placed at the center
<code>justify-between</code>	<code>justify-content: space-between;</code>	All columns are spaced out as much as possible with first column at the beginning and last column at the end (We just saw this in action)
<code>justify-around</code>	<code>justify-content: space-around;</code>	Space <i>before</i> the columns and <i>after</i> the columns are half as much as space between the columns
<code>justify-evenly</code>	<code>justify-content: space-evenly;</code>	Space before, after and between the columns are same

By default, if the sum total width of all items are smaller than the space available in the grid container, the `auto` sized elements' widths are increased equally to fill the container. This is exactly what's happening in our example before we add the `justify-between` class.

In your browser (Chrome or Firefox), use the inspector tool on the Tailwind Play link above and click on "grid" next to `<div class="container">` `class="container">` `grid` :

This will highlight the grid cells. Before setting the `justify-content` property, this is what we see:



Using `grid-cols-*` we have set our columns to `auto` size. Since the widths of all logos combined is smaller than the width of the container, the remaining space is equally divided and added to each column. (This is not the same as using the `fr` unit. If you use `1fr`, each column ends up with the same width).

But when we add `justify-between`, this happens:

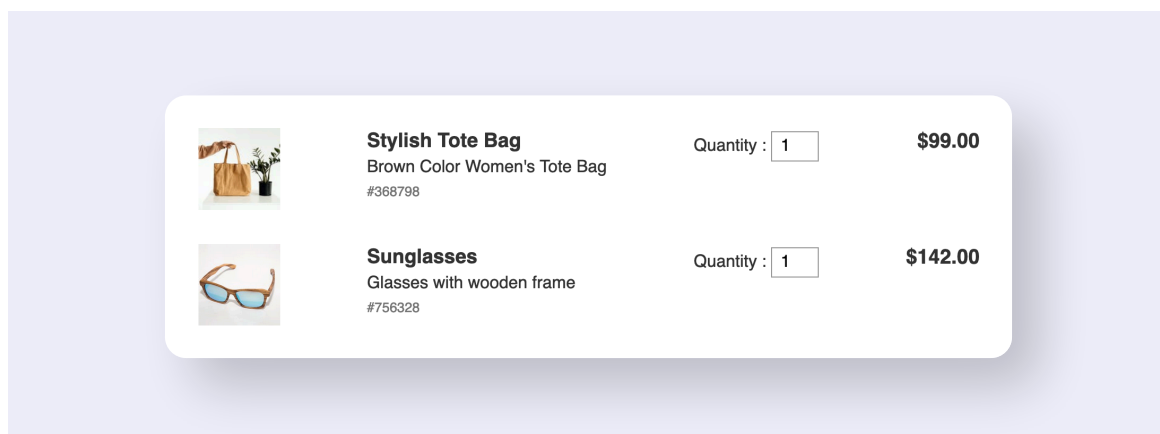


The grid cells now take up only as much width required. The remaining space is added to the grid lines (or grid gaps) instead of the grid cells.

**Note :** If we use the `fr` units for specifying the width for any of the grid columns, there is no space remaining and hence the utilities `justify-*` will have no effect! Which is why we used `grid-cols-[repeat(4, auto)]` instead of `grid-cols-4`.

## Shopping Cart Summary Example 19b

Here's another great example for CSS Grid - a shopping cart summary with each row containing an image, product description, quantity and price. You can use `justify-*` here too, to space out the columns.





## ▶ Try it out

### Markup

```
1 <div class="container grid grid-cols-[repeat(4,auto)] gap-y-8 gap-x-4">
2   <img ... >
3   <div class="desc"> ... </div>
4   <div class="qty"> ... </div>
5   <div class="price"> ... </div>
6   ...
7 </div>
```

### Solution

Along with using `justify-between` property, we also need to add a `text-right` to the `.price` element to align the text in the last column to right.

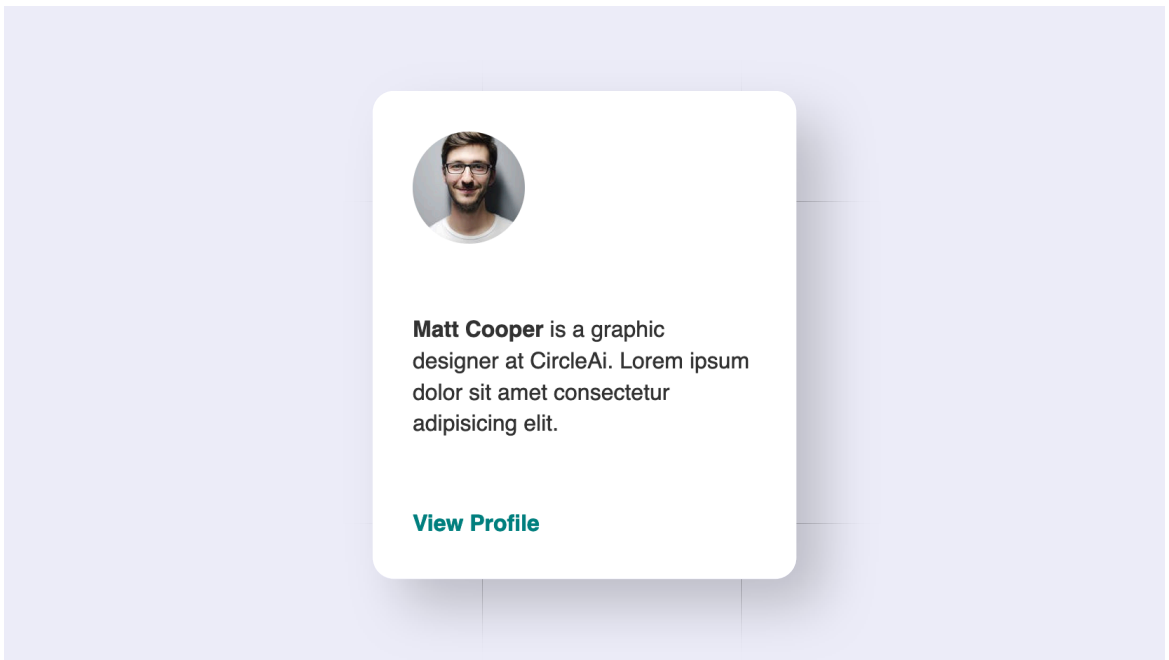
```
1 <div class="container grid grid-cols-[repeat(4,auto)] gap-y-8 gap-x-4 justify-between">
2   ...
3   <div class="price text-right"> ... </div>
4   ...
5   <div class="price text-right"> ... </div>
6 </div>
```

## ▶ Working Demo

## 20 Align Content

### Profile Card with Bio & Link Example 20a

Assume you need a profile card with a fixed height containing the profile picture, short bio and a link - all three elements spaced out equally. This can be achieved using flexbox too, but the solution with grid is one utility class lesser.



▶ Try it out

### Markup

```
1 <div class="card h-96 grid">
2   <img ... />
3   <p> ... </p>
4   <a> ... </a>
5 </div>
```

## Solution

For this one, we need only one column, so adding `grid` to the `.card` element automatically creates a grid with one column and three rows with `auto` heights. Now we just need to control the vertical spacing using `content-*` utilities.

```
1 <div class="card h-96 grid content-between">
2   ...
3 </div>
```

▶ Working Demo

## Understanding Align Content in Grid Concept

We have seen `content-*` utilities [with respect to flexbox](#) to control spacing between multiple lines of wrapped items along the *cross axis*. In Grid, these are used to control spacing between the rows. `content-between` is one of the available utilities we just used. The available utilities are similar to that of `justify-*`.

The `justify-*` utilities control placement of columns within container, while `content-*` utilities control placement of rows within the container.

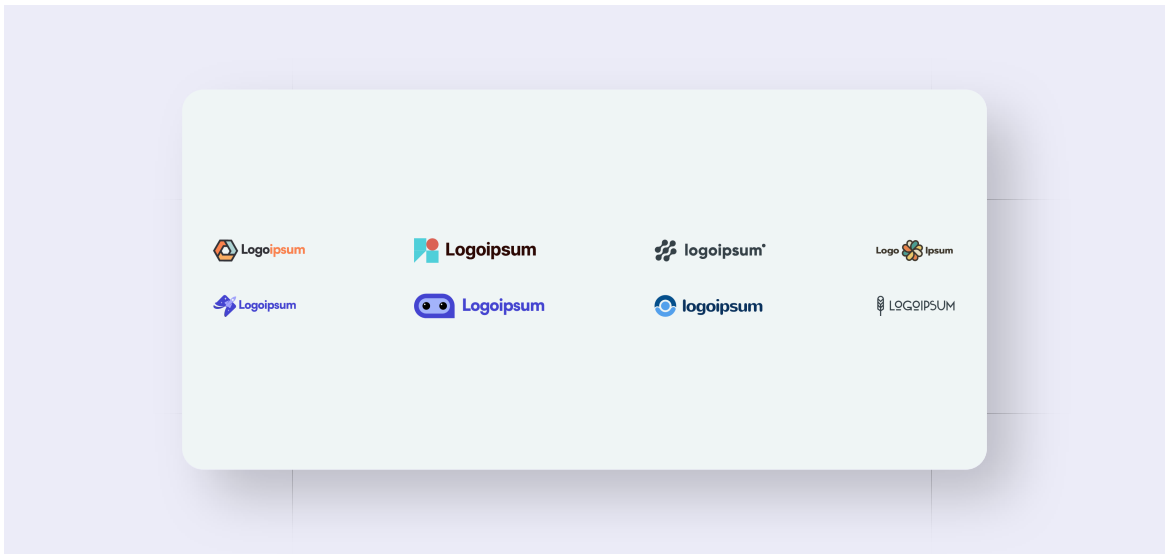
Tailwind Class	CSS Property & Value	Explanation
<code>content-start</code>	<code>align-content: flex-start;</code>	All rows are placed at the beginning of the container
<code>content-end</code>	<code>align-content: flex-end;</code>	All rows are placed at the end of the container
<code>content-center</code>	<code>align-content: center;</code>	All rows are placed at the center
<code>content-between</code>	<code>align-content: space-between;</code>	All rows are spaced out as much as possible with first row at the beginning and last row at the end (We just saw this in action)
<code>content-around</code>	<code>align-content: space-around;</code>	Space <i>before</i> the rows and <i>after</i> the rows are half as much as space between the rows
<code>content-evenly</code>	<code>align-content: space-evenly;</code>	Space before, after and between the rows are same

**Note :** These utilities have an effect only when

1. The `grid` container has a `height` value that is greater than the sum of individual row heights
2. **And** when none of the grid items has a height specified in `fr` units

## Featured Logos Center of Page Example 20b

In [Example 19a](#), we looked at displaying logos in a grid and spacing them horizontally with `justify-between`. Now what if we want both the rows to be at the center of the page vertically?



Note that as soon as you add a height of `100vh` to the container, each row stretches to occupy 50% of the height each and hence the logos are spread out vertically. We need to bring them closer and place them at the center. Yes, this can be done with `content-*` utilities because this controls the placement of the rows.

▶ Try it out

### Markup

```
1 <div class="container min-h-screen grid grid-cols-[repeat(4,auto)]
  gap-12 justify-between">
2   <img ... >
3   <!-- Seven more img elements -->
4 </div>
```

### Solution

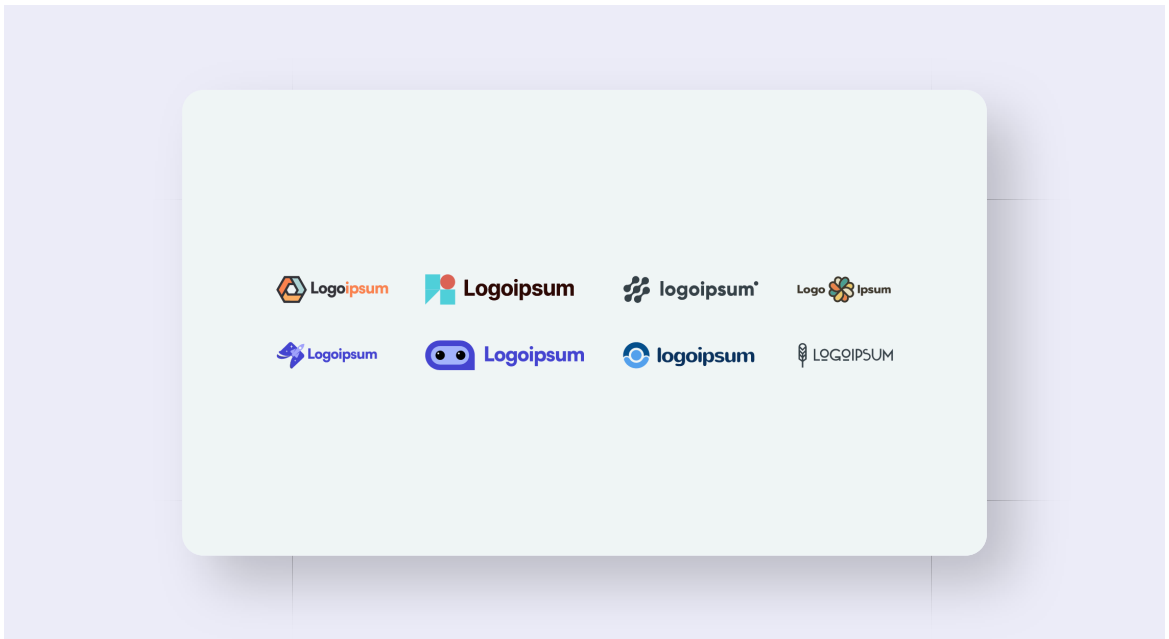
There are surely other ways of doing it, but the simplest solution is to use `content-center` along with `justify-between`.

```
1 <div class="container min-h-screen grid grid-cols-[repeat(4,auto)]
  gap-12 justify-between content-center">
2   ...
3 </div>
```

## ▶ Working Demo

### A small variation

Now assume you don't want the logos to space out horizontally, instead you want them to be centered horizontally too.



### Solution

For this, you can use `justify-center` instead of `justify-between`. So now we have:

```
1 <div class="container ... justify-center content-center">
2   ...
3 </div>
```

## Better Solution

```
1 <div class="container ... place-content-center">
2   ...
3 </div>
```

We have combined `justify-center` and `content-center` into `place-content-center`.

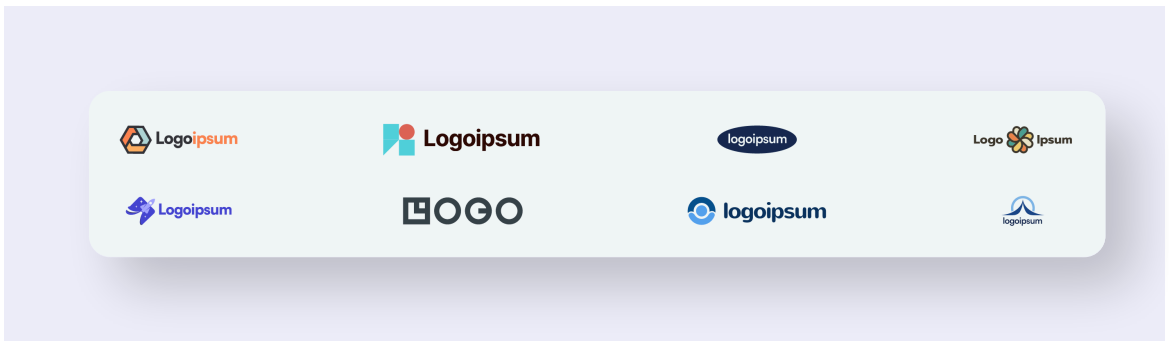
## Understanding Place Content in Grid Concept

The `place-content-*` utilities allows you to control the spacing of grid items along both the block and inline axes at once. But this is possible only when you want the placement of rows and columns to be the same. In the previous example, we wanted the rows AND columns to be placed at the center of the grid container. Hence we could use the `place-content-*` utilities. The available utilities are similar to that of `justify-*` and `content-*`.

## 21 Justify Items

### Featured Logos of Different Widths Example 21a

We're back with the same example with yet another variation. Previously all the logos we used were approximately of the same dimensions, hence it looked good. But if you add some smaller or wider logos, you need to center align the logos in each column.



In the below link, you can see that the wider logos increase the width of the columns and by default, the smaller ones are aligned to the left of those columns. Use the inspector tool to take a closer look at what's happening. How can we center those smaller logos within the columns?

▶ Try it out

### Markup

```
1 <div class="container grid grid-cols-[repeat(4,auto)] gap-12 justify-  
  between">  
2   <img ... >  
3   <!-- Seven more img elements -->  
4 </div>
```

### Solution

There's one new utility class to our rescue - `justify-items-*`.



```
1 <div class="container grid grid-cols-[repeat(4,auto)] gap-12 justify-
  between justify-items-center">
2   ...
3 </div>
```

▶ Working Demo

## Understanding Justify Items Concept

As we now know, CSS Grid creates something similar to a table with rows and columns. If you have experience with tables (or even simple spreadsheets), you know that adding more content to any one cell widens that entire column. Same thing happens here too. And while that happens, all the other cells in that column will have content sticking to the left of that column by default.

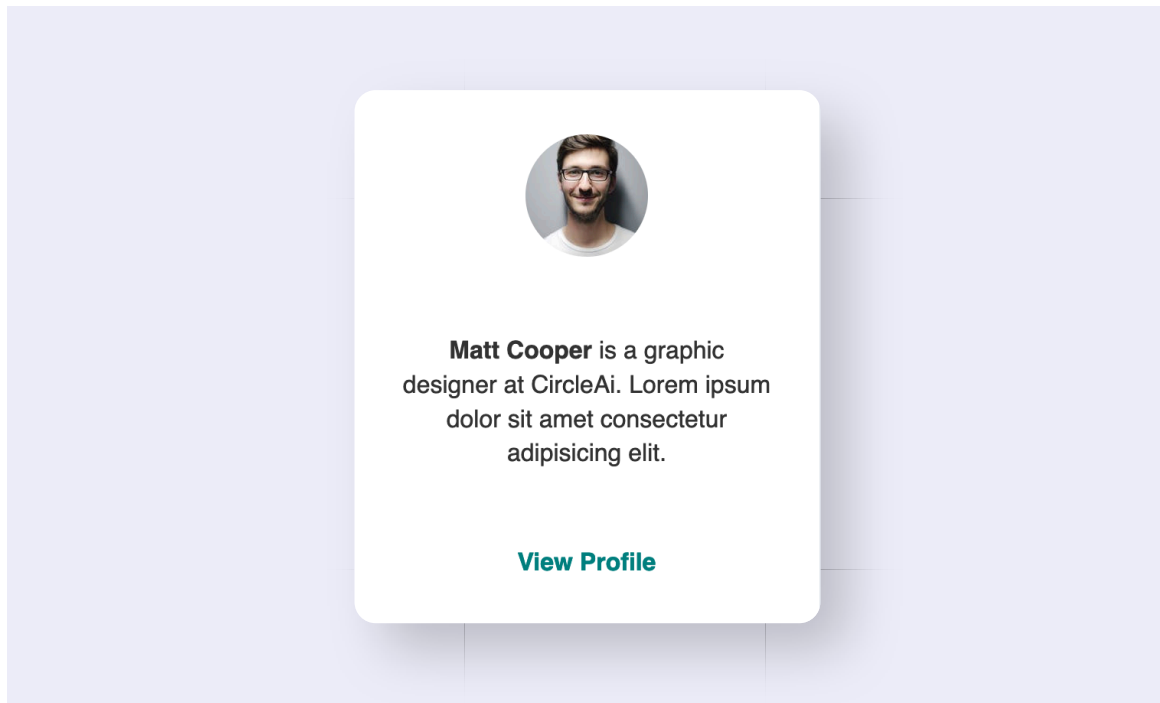
The `justify-items-*` utilities allows us to **horizontally** align the content *within* the columns, while the previous utilities `justify-*` allows us to control spacing of the entire columns. The available utilities in Tailwind are:

Tailwind Class	CSS Property & Value	Explanation
<code>justify-items-start</code>	<code>justify-items: start;</code>	All items are placed at the beginning of their columns (horizontally)
<code>justify-items-end</code>	<code>justify-items: end;</code>	All items are placed at the end of their columns (horizontally)
<code>justify-items-center</code>	<code>justify-items: center;</code>	All items are placed at the center of their columns (horizontally)
<code>justify-items-stretch</code>	<code>justify-items: stretch;</code>	The items are stretched to occupy full width of the column if possible

**Note :** This property does not make sense with flexbox because the elements are laid out and aligned in only one direction.

## Profile Card with Bio & Link Centered Example 21b

We are revisiting [Example 20a](#), this time making everything center aligned horizontally.



▶ Try it out

Yes, this is possible with `mx-auto` and `text-center` applied to grid items individually. But best solution is to apply styles to the container directly.

## Markup

```
1 <div class="card h-96 grid content-between">
2   <img ... >
3   <p> ... </p>
4   <a> ... </a>
5 </div>
```

## Solution

We can use the utility `justify-items-center` on the container to horizontally center smaller and wider items within each column.

```
1 <div class="card h-96 grid content-between justify-items-center text-
  center">
2   ...
3 </div>
```

We additionally need a `text-center` to center align the paragraph.

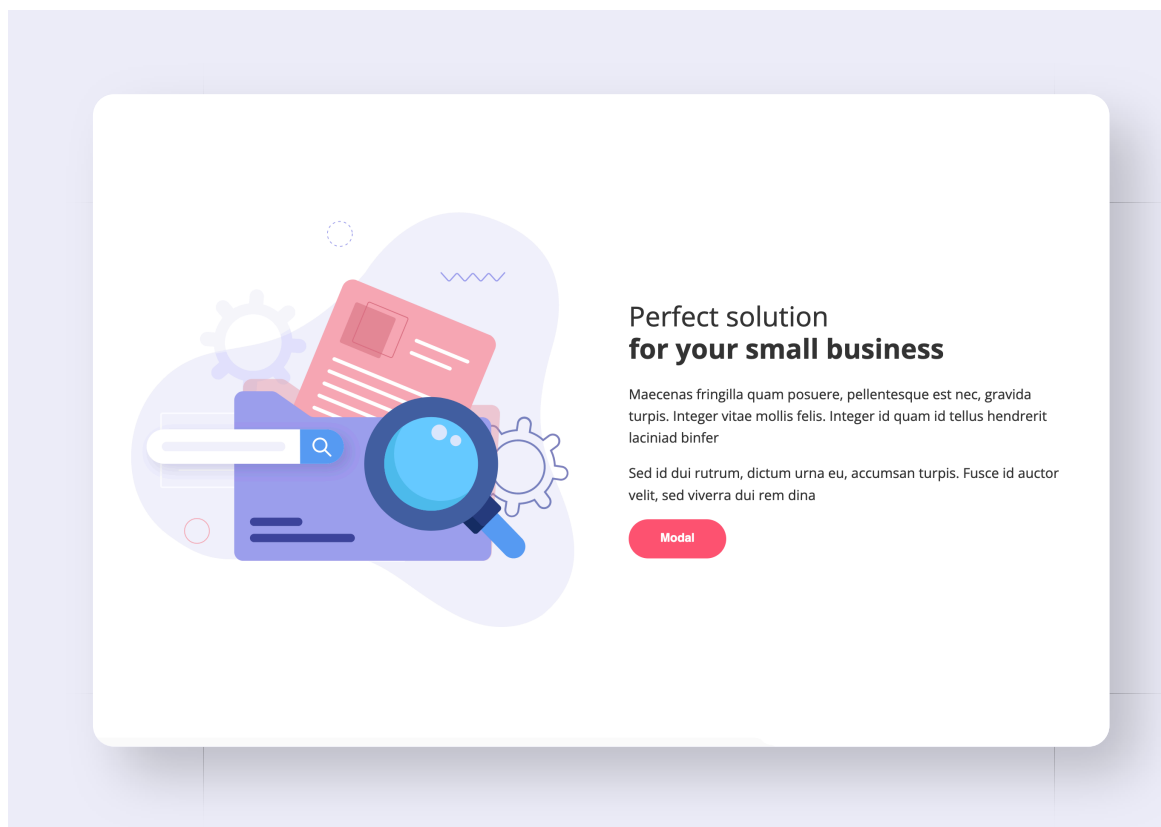
▶ [Working Demo](#)

## 22 Align Items

### Image and Text Section Example 22a

Example from [Inovatik Template](#)

One very common section in web pages is an image on the left half and text on right half of the page. You need the text and image to be perfectly center aligned vertically for all large screen sizes. Grid is great for something like this.



► Try it out

## Markup

```
1 <section class="container min-h-screen grid grid-cols-2 gap-16">
2   <img ... >
3   <div> ... </div>
4 </section>
```

## Solution

Using `grid-cols-2` we have created two equal sized columns. Using `gap`, we have added some spacing between them. To center align the image and text vertically in the page, we need to use the `items-*` utility, very similar to flexbox.

```
1 <section class="container min-h-screen grid grid-cols-2 gap-16 items-
  center">
2   ...
3 </section>
```

### ▶ Working Demo

The benefit of using grid over flexbox for this example is that `gap` property for grid is supported in more browsers than for flexbox. Everything else is quite similar.

## Understanding Align Items in Grid Concept

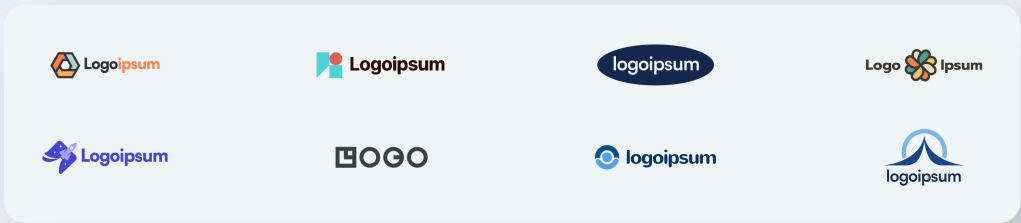
In grid, we observed earlier that adding more content to any one cell widens that entire column **and** increases the height of that row too. And while that happens, all the other cells in that row will have content sticking to the top of that column by default. Which is why, in our previous example, when the image is taller than the text, the text is aligned to the top of the row and when the text is taller than the image, the image is aligned to the top.

The `items-*` utilities allow us to **vertically** align the content *within* the rows, while the previous property `content-*` allowed us to control spacing of entire rows. We have already seen the available utilities when we covered [this concept with respect to flexbox](#). Here they are again, for reference.

Tailwind Class	CSS Property & Value	Explanation
<code>items-stretch</code>	<code>align-items: stretch;</code>	All items are stretched to fill the container
<code>items-center</code>	<code>align-items: center;</code>	All items are aligned to the center of the container
<code>items-start</code>	<code>align-items: flex-start;</code>	All items are aligned to the beginning of the container ( <i>at the top in case of the above example</i> )
<code>items-end</code>	<code>align-items: flex-end;</code>	All items are aligned to the end of the container ( <i>at the bottom in case of the above example</i> )
<code>items-baseline</code>	<code>align-items: baseline;</code>	All items are positioned such that the base aligns to the end of the container ( <i>will we talk about this soon</i> )

## Featured Logos of Different Heights Example 22b

Let's look at the same example once again. So far, we used a fixed height `h-10` for all the logos. Now we'll remove that and instead add a `max-w-[10rem]` (Check the CSS tab). So now, all the logos have different heights and different widths too. The way we horizontally center aligned the logos in each column in [Example 21a](#), this time we also need to vertically center align them in each row.



▶ Try it out

### Markup

```
1 <div class="container ... justify-between justify-items-center">
2   <img ... >
3   <!-- Seven more img elements -->
4 </div>
```

### Solution

We can use the utility `items-center` to vertically center the taller and shorter logos within each row

```
1 <div class="container ... justify-between justify-items-center items-
   center">
2   ...
3 </div>
```

▶ Working Demo

## Summary

`justify-between` is to space out the entire rows horizontally

`justify-items-center` is to center align smaller and wider logo horizontally within each column

`items-center` is to center align shorter and taller logos vertically within each row



## 23 Place Items

### Center a div using Grid

Example 23a

We have already seen how easy it is to [center a div using flexbox](#). With grid, it's one lesser utility class.



### Markup

```
1 <div class="container">
2   <div class="item">
3     ...
4   </div>
5 </div>
```

### Solution

You can either use the previous two utilities `justify-items-center` and `items-center` along with `grid`. Or you can combine both these using the `place-items-*` utilities.

```
1 <div class="container grid place-items-center">
2   <div class="item">
3     ...
4   </div>
5 </div>
```

▶ Working Demo

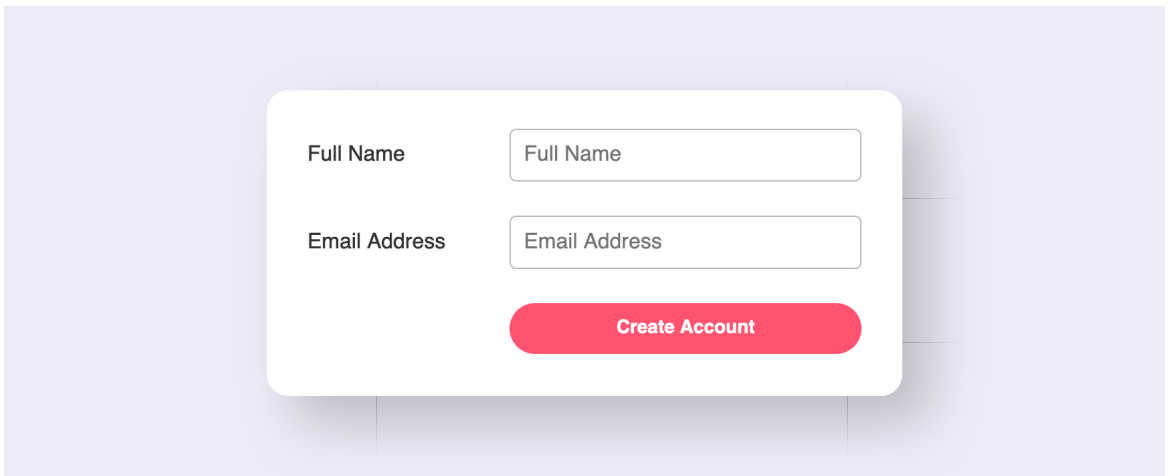
## Understanding Place Items Concept

The `place-items-*` utilities allows you to align the items horizontally within columns and vertically within rows at once. But this is possible only when you want the same alignment in both directions. In the previous example, we wanted the item to be at the center horizontally and vertically. Hence we could use the `place-items-*` utilities. The available utilities are similar to that of `justify-items-*` and `items-*`.

## 24 Grid Column Start, End & Span

### Horizontal Form Example 24a

Creating forms and making them responsive is so much more easier with grid than any other tool. Here's the simplest example of a horizontal form with labels on the left, inputs on the right and a button on the right too. With the knowledge of Grid so far, you can surely create this component.



We can use `grid-cols-[auto, 1fr]` to create two columns as required. But since the button "Create Account" is the 5th item in the markup, it appears on the first column. How do we make it appear on the second column instead?

▶ Try it out

## Markup

```
1 <form class="grid grid-cols-[auto,1fr] items-center gap-y-6 gap-x-12">
2   <label></label>
3   <input .. />
4   <label></label>
5   <input .. />
6   <button ...>Create Account</button>
7 </form>
```

## Solution

Of course, one solution is to add a dummy element in HTML before the button, but that's definitely not a good practice. And there's a simple utility class available for this:

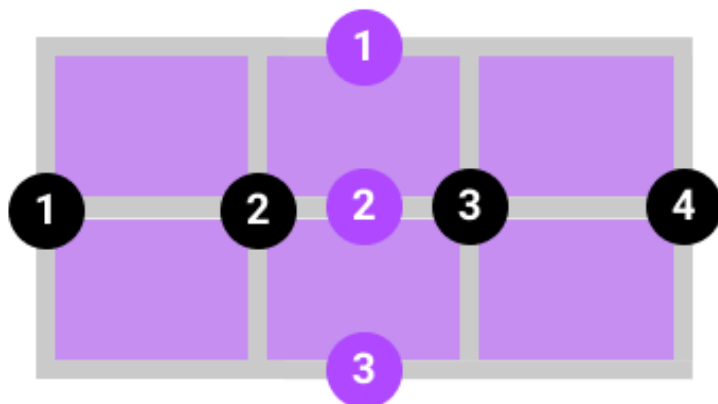
```
1 <form class=" ... ">
2   ...
3   <button class="col-start-2" ...>Create Account</button>
4 </form>
```

### ▶ Working Demo

We have used `col-start-2` on the **grid item** to change the column it appears in. Let's learn about this.

## Understanding Column Start Concept

Before we learn more about the `col-start-*` utilities, we need to learn about **grid lines**. When you define a grid using `grid-cols-*` and/or `grid-rows-*`, grid lines are created. These are nothing but the lines between and around the columns and rows. This picture explains how the column lines and row lines are numbered.



It's important to remember that the line numbers **start from 1** and not 0.

All the grid related utilities we saw so far are applied to the grid container. Now, we will see a few that are applied to the grid items. The `col-start-*` is one of them. It specifies the item's start position. Some of the available Tailwind utilities for this are:

Tailwind Class	CSS Property & Value	Explanation
<code>col-start-1</code>	<code>grid-column-start: 1;</code>	The item starts at column line 1
<code>col-start-2</code>	<code>grid-column-start: 2;</code>	The item starts at column line 2

Such utilities are available upto `col-start-13`.

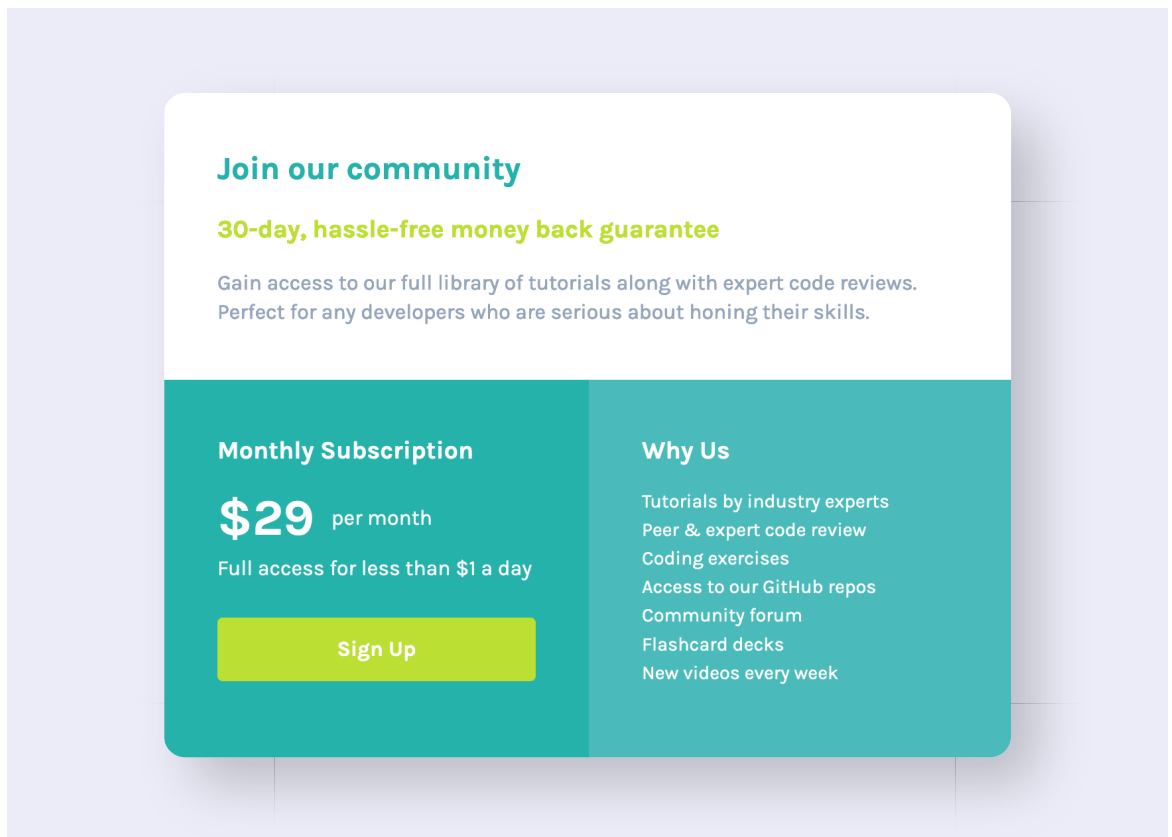
In CSS, you can also use negative values like `grid-column-start: -1`. Here `-1` specifies the last column line. When negative integers are used, it starts counting the lines in reverse starting from `-1`.

In the above example, we `col-start-2` because we wanted the button to start from column line 2. Now observe that if you use `col-start-3`, it creates a new column and everything gets messed up. Hence you need to be careful with this utility class.

## Single Price Grid Component Example 24b

Challenge from [Frontend Mentor](#)

This grid component is a challenge from the Frontend Mentor website. This is responsive with the mobile version having just one column with all three items one below the other. The desktop version however has two columns, but the first item spans across both the columns.



For smaller screens, nothing needs to be done. For larger screens, you can make the first item span using the `col-start-*` utility that we just saw along with `col-end-*`.

▶ Try it out

## Markup

```
1 <div class="container grid sm:grid-cols-2">
2   <div class="component-header"> ... </div>
3   <div class="subscription"> ... </div>
4   <div class="why"> ... </div>
5 </div>
```

## Solution

Above the `sm` breakpoint, we need to make the `.component-header` start at column line 1 and end at column line 3.

```
1 <div class="container grid sm:grid-cols-2">
2   <div class="component-header sm:col-start-1 sm:col-end-3"> ... </div>
3   ...
4 </div>
```

### ▶ Working Demo

Let's learn about `col-end-*` utilities and few more ways of getting the same result.

## Understanding Column End Concept

The utilities `col-end-*` is another set of grid item's utilities. It specifies the item's end position. The Tailwind utilities available for this are similar to `col-start-*`. In CSS, you can also use a negative integer for `grid-column-end`, in which case it starts counting the lines in reverse, starting from `-1`.

In the above example, we used `col-end-3` because we wanted that item to end at column line 3. You can also use arbitrary value `col-end-[-1]`.

We can alternatively use another set of utilities `col-span-*` which can be used to specify the number of columns to span. This is usually used along with either `col-start-*` or `col-end-*`.

So, another solution to the previous example can be:

```
1 <div class="component-header sm:col-start-1 sm:col-span-2"> ... </div>
```

This means, start from column line 1 and span across two columns. This is helpful when you know where to start and how many columns you want the item to span, but don't want to calculate the end line.

You can also skip the `col-start-1` here:

```
1 <div class="component-header sm:col-span-2"> ... </div>
```

Because the starting column line is 1 by default.

OR even

```
1 <div class="component-header sm:col-span-full"> ... </div>
```

OR

```
1 <div class="component-header sm:col-span-2 sm:col-end-3"> ... </div>
```

This means, end at column line 3 by spanning across two columns. Again this is helpful when you know where to end and how many columns to span, but don't want to calculate the start line. Don't miss trying out all these in the previous two examples.

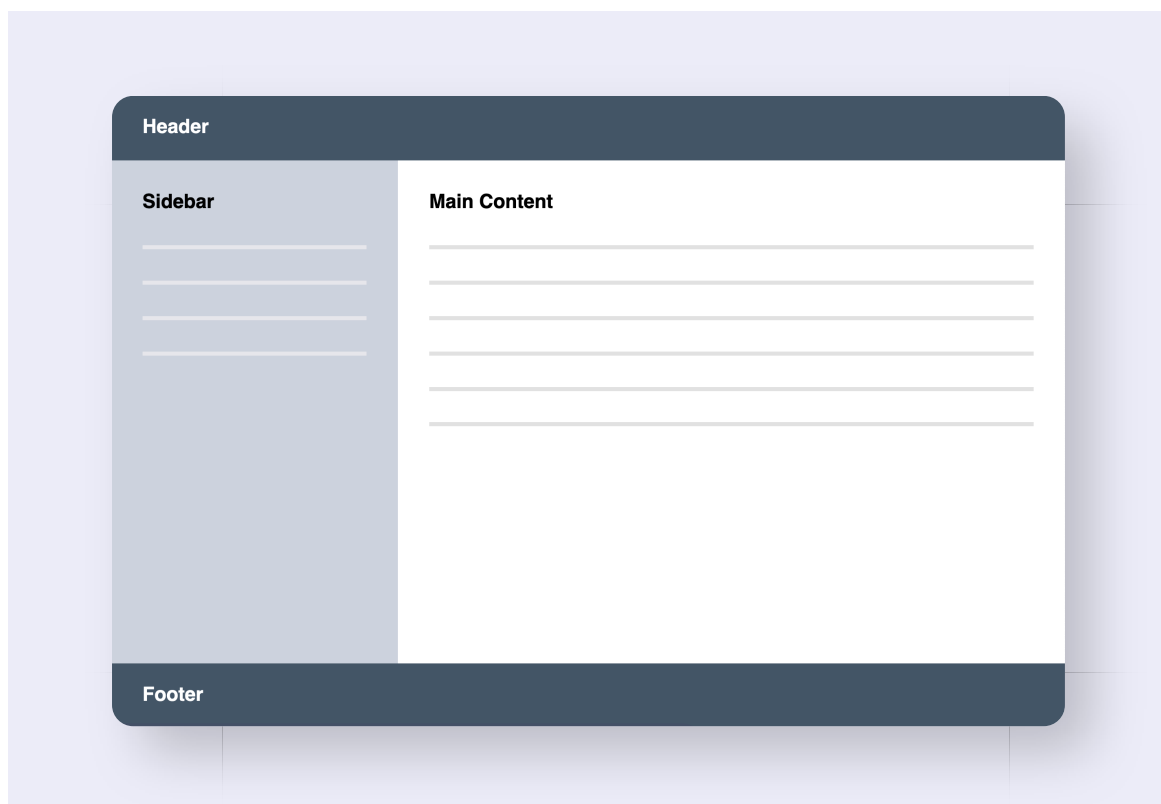


# Understanding Column Span Concept

The `col-span-*` utilities can be used on a grid item to specify how many columns to span. This is usually used along with either `col-start-*` or `col-end-*`. But if used alone, the default start and end lines are considered. The available utilities in Tailwind are `col-span-1` upto `col-span-12` along with a helpful `col-span-full` which makes the grid item span across all the columns in the grid.

## Page Layout with Grid Example 24c

We already saw how to implement a [layout with sidebar](#) and [sticky footer using grid](#), which are the simplest examples of creating full page layouts using grid. Let's look another layout that combines the above two with a header, sidebar, main content and a footer.



This is a grid layout with 2 columns and 3 rows. The `header` and `footer` span across both the columns.

▶ Try it out

## Markup

```
1 <div class="container min-h-screen">
2   <header> ... </header>
3   <div class="sidebar"> ... </div>
4   <div class="main"> ... </div>
5   <footer> ... </footer>
6 </div>
```

## Solution

There are three things we need to do:

1. We have a fixed width sidebar, so the `grid-cols-*` will have one fixed width value and `1fr`.
2. We will have to use `grid-rows-*` to control the sizing of the rows because we need the header and footer to occupy only as much as the content within, so we use `auto` for header and footer. And we want the middle row to occupy as much height as possible, so we use `1fr`.
3. And for the header and footer, we use `col-span-*` utilities to span two columns.

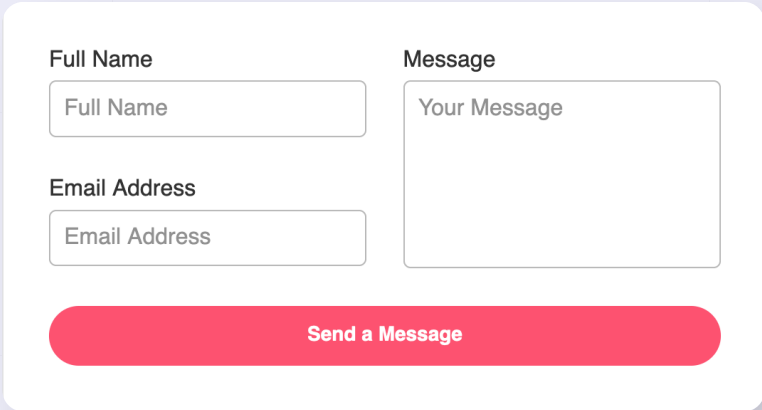
```
1 <div class="container min-h-screen grid grid-cols-[22rem,1fr] grid-rows-[auto,1fr,auto]">
2   <header class="col-span-full"> ... </header>
3   ...
4   <footer class="col-span-full"> ... </footer>
5 </div>
```

▶ Working Demo

## 25 Grid Row

### Contact Form Example 25a

Let's look a contact form with a couple of fields in the first column and one field in the second column. Previously we saw grid items spanning across columns, but here one grid item spans across rows too. The concept is the same.



The image shows a contact form with a grid layout. The form is white with rounded corners and a shadow. It has two columns. The first column has two input fields: "Full Name" and "Email Address". The second column has a larger text area labeled "Message". At the bottom, there is a red button labeled "Send a Message".

▶ Try it out

### Markup

```
1 <form class="grid grid-cols-2 gap-6">
2   <div>
3     <label>... </label>
4     <input ... />
5   </div>
6   <div>
7     <label>... </label>
```

```

8     <input ... />
9   </div>
10  <div class="message-block">
11    <label> ... </label>
12    <textarea> ... </textarea>
13  </div>
14  <button> ... </button>
15 </form>

```

## Solution

```

1 <form class="grid grid-cols-2 gap-6">
2   ...
3   <div class="col-start-2 row-start-1 row-end-3">
4     <label> ... </label>
5     <textarea> ... </textarea>
6   </div>
7   <button class="col-span-full"> ... </button>
8 </form>

```

### ▶ Working Demo

We have already learned about the `col-start` utilities. Now we're using two new utilities `row-start-*` and `row-end-*`.

## Understanding Row Start and Row End Concept

The utilities `row-start-*` and `row-end-*` are also grid item's properties. `row-start-*` specifies the item's start position and `row-end-*` specifies the item's end position with respect to row lines.

In the previous example, we want the message block to start from row line 1 and end at row line 3, apart from starting at column line 2. It's important to specify all these row and column lines.

The available utility classes in Tailwind are similar to that of `col-start-*` and `col-end-*`.

## Understanding Row Span Concept

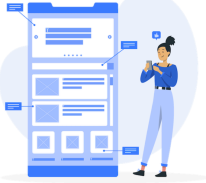
Similar to `col-span-*` utilities, we also have `row-span-*` utilities for grid items to specify how many rows to span. This is usually used along with either `row-start-*` or `row-end-*`. But if used alone, the default start and end lines are considered. The available utilities in Tailwind are `row-span-1` upto `row-span-12` along with a helpful `row-span-full` which makes the grid item span across all the columns in the grid.

**Note:** It's always better to use the start and end utilities instead of span utilities.

## Responsive Services Section Example 25b

Example Inspired from [Brian Haferkamp's CodePen](#) & [User illustrations by Storyset](#)

This is a section with an image and six services presently differently in three different screen sizes. Mobile layout has one grid column, tablet layout has two grid columns and desktop layout has three grid columns. But more importantly, the grid items' placements change. This is quite simple now using `col-start`, `col-end`, `row-start` and `row-end`.



#### List Building

It's very easy to start creating email lists for your marketing actions, give it a try

#### Campaign Tracker

Campaigns is a feature we've created since the beginning and it's at the core of Lomar

#### Analytics Tool

Lomar collects all the necessary data to help you analyse different situations

#### Admin Control

Rights of users and admins can easily be managed through the control panel



#### List Building

It's very easy to start creating email lists for your marketing actions, give it a try

#### Campaign Tracker

Campaigns is a feature we've created since the beginning and it's at the core of Lomar

#### Analytics Tool

Lomar collects all the necessary data to help you analyse different situations

#### Admin Control

Rights of users and admins can easily be managed through the control panel

#### Integration Setup

We're providing a step-by-step integration session to implement automation

#### Help Line Support

Quality support is our top priority so please contact us for any problem you encounter

#### List Building

It's very easy to start creating email lists for your marketing actions, give it a try

#### Campaign Tracker

Campaigns is a feature we've created since the beginning and it's at the core of Lomar

#### Analytics Tool

Lomar collects all the necessary data to help you analyse different situations



#### Admin Control

Rights of users and admins can easily be managed through the control panel

#### Integration Setup

We're providing a step-by-step integration session to implement automation

#### Help Line Support

Quality support is our top priority so please contact us for any problem you encounter

## ▶ Try it out

### Markup

```
1 <section class="grid sm:grid-cols-2 md:grid-cols-3 gap-x-8">
2   <img ... >
3   <div> ... </div>
4   <div> ... </div>
5 </section>
```

### Solution

Following mobile-first approach, we start with one single column, change to two columns at `sm` breakpoint and to three columns at `md` breakpoint.

```
1 <section class="grid sm:grid-cols-2 md:grid-cols-3 gap-x-8">
2   <img class="sm:row-start-1 sm:row-end-3 md:row-end-2 md:col-start-2"
3   ... >
4   <div class="md:col-start-1 md:text-right">
5     ...
6   </div>
7   <div class="sm:col-start-2 md:col-start-3">
8     ...
9   </div>
10 </section>
```

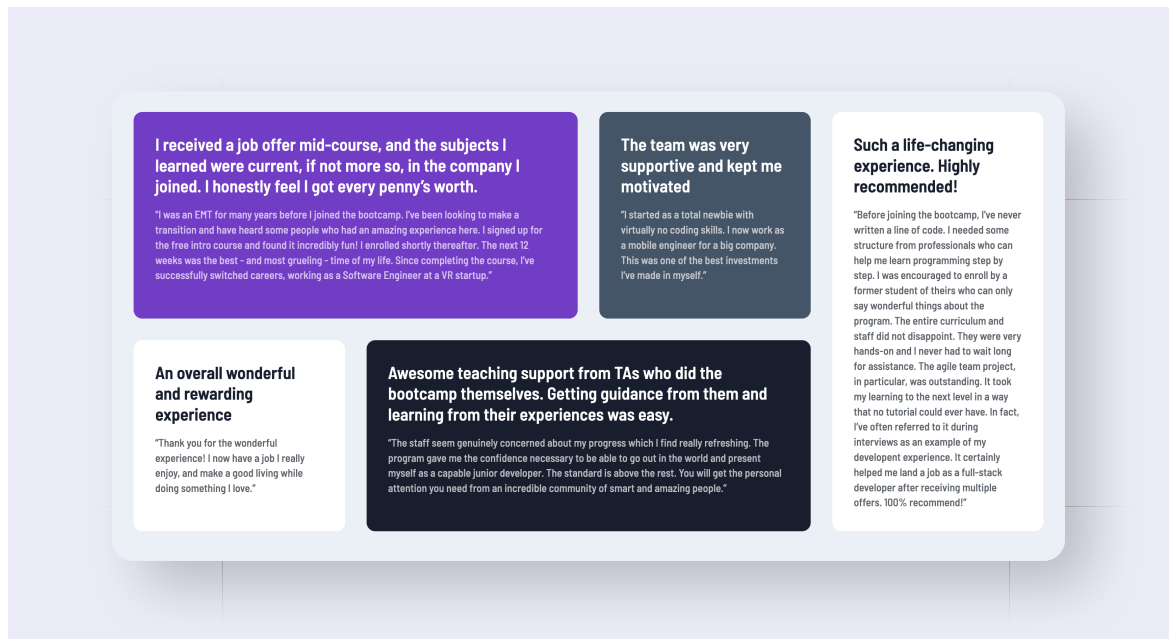
We have changed the row and column lines for `img` and both the divs at `sm` and `md` breakpoint. Look at the working demo and carefully observe how it's done.

## ▶ Working Demo

## Testimonials Grid Section Example 25c

Challenge from [Frontend Mentor](#)

Here's another example from Frontend Mentor website, but with removed avatars and names for simplicity. Go for the mobile first approach with just one column and above 1g breakpoint, try and achieve this layout.



▶ Try it out

### Markup

```
1 <section>
2   <div class="violet"> ... </div>
3   <div class="gray"> ... </div>
4   <div class="white"> ... </div>
5   <div class="dark"> ... </div>
6   <div class="white-long"> ... </div>
7 </section>
```



## Solution

On mobile, you just have to apply `grid` and `gap-8` to `section` and everything just works. Above the `lg` breakpoint, you need to create four columns and almost every grid item needs to be positioned using the row and column lines.

```
1 <section class="grid lg:grid-cols-4 gap-8">
2   <div class="violet lg:col-span-2"> ... </div>
3   <div class="gray"> ... </div>
4   <div class="white lg:row-start-2"> ... </div>
5   <div class="dark lg:col-span-2"> ... </div>
6   <div class="white-long lg:row-start-1 lg:row-span-2 lg:col-start-
   4"> ... </div>
```

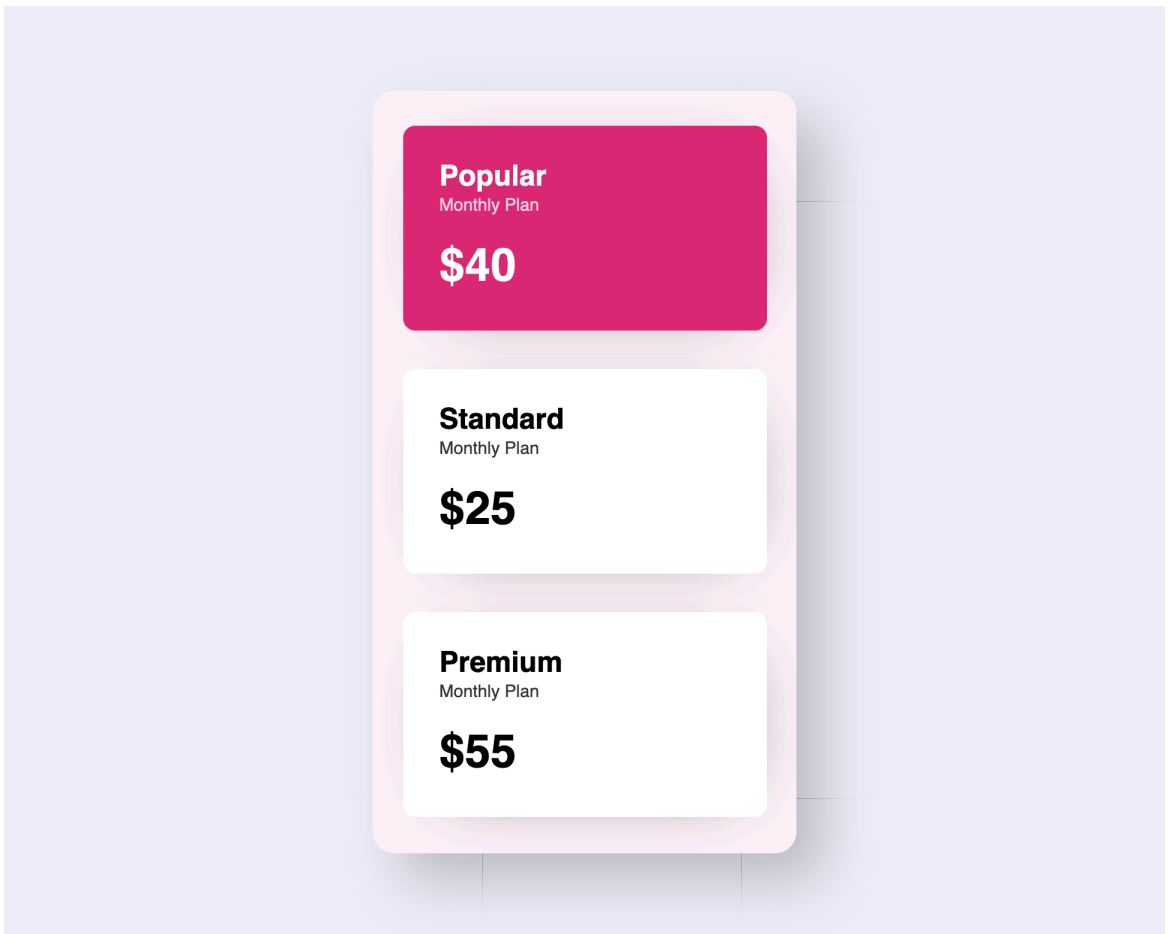
Most important is positioning of that `.white-long` div. You need to mention both the row lines along with the column line. Here, I have used `row-span-2`, but you can also use `row-end-3`.

▶ Working Demo

## 26 Order

### Responsive Pricing Plans Example 26a

Let's look at the [Pricing Plans Example](#) again and make it responsive with one change. On mobile screens, we place the **Popular** plan first, followed by **Standard** and **Premium** while keeping the order same on desktop.



▶ Try it out

One way to approach this is to change the column lines using `col-start` for mobile and change it back for larger screens. That surely works, but too confusing. Let's look at another solution using the `order` property.

## Markup

```
1 <div class="container grid sm:grid-cols-3 gap-8">
2   <div class="plan"> ... </div>
3   <div class="plan plan-highlight"> ... </div>
4   <div class="plan"> ... </div>
5 </div>
```

## Solution

The `.plan-highlight` element is the popular plan that we want to place first on mobile screens.

```
1 <div class="plan plan-highlight order-first sm:order-none"> ... </div>
```

### ▶ Working Demo

On mobile screens, we are placing the popular plan first using `order-first` and at `sm` breakpoint, we are changing back to default order using `order-none`.

## Understanding Order in Grid Concept

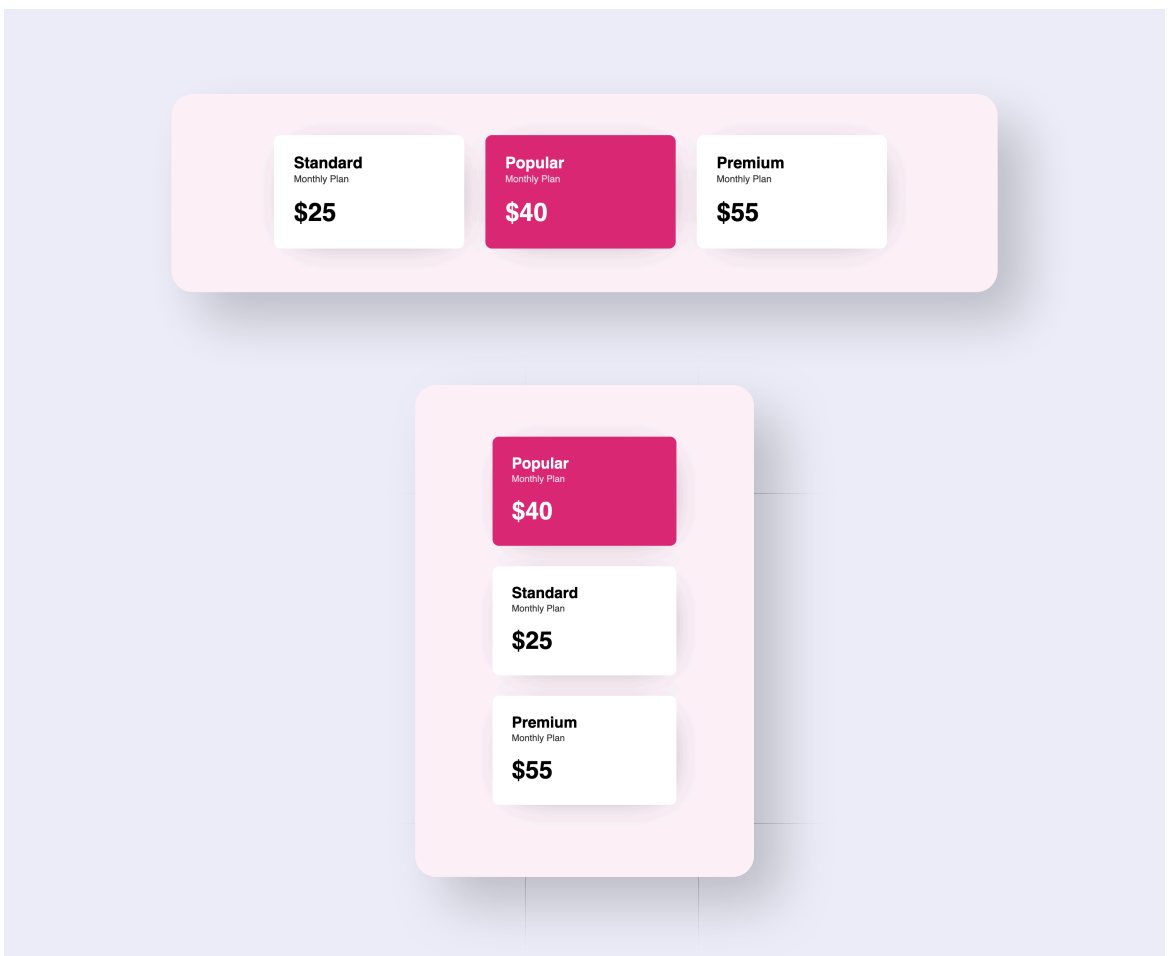
The same `order` utilities that we saw [with respect to flexbox](#) can be used for grid items too. The value can be any number - positive or negative. The items with greater `order` value appear later on the web page compared to the items with lesser value irrespective of their appearance in the markup.

If no `order` is specified, by default the value is `0` for all the elements and they follow the same order as they appear in HTML. That's what happens at `sm` breakpoint.

## 27 Advanced Grid Template Values

### Pricing Plans with Size Limits Example 27a

In the pricing plans example we saw earlier, you might have noticed that the columns stretch full width of the container even on mobile screens. But since our content within each card is too small, a wide card looks bad. So we want to limit the width of each card to a maximum of say `18rem` and also to a minimum of the content within so that it doesn't shrink below that width.



## ▶ Try it out

### HTML

```
1 <div class="container grid sm:grid-cols-3 gap-8">
2   <div class="plan"> ... </div>
3   <div class="plan plan-highlight"> ... </div>
4   <div class="plan"> ... </div>
5 </div>
```

### Solution

There is no pure Tailwind solution for this. We can add a `max-w-` and `min-w-` to the `.plan` element itself to limit the width of the cards. But then the column width still remains large which makes it impossible to center align the three cards together on larger screens. So, here's the perfect solution:

```
1 <div class="container
2     grid
3     grid-cols-[minmax(auto,18rem)]
4     sm:grid-cols-[repeat(3,minmax(auto,18rem))]
5     justify-center
6     gap-8">
7   ...
8 </div>
```

## ▶ Working Demo

We have replaced `grid-cols-3` with `grid-cols-[repeat(3,minmax(auto, 18rem))]`.

In CSS, this is same as:

```
grid-template-columns: repeat(3, minmax(auto, 18rem))
```

## Understanding `minmax()` Concept

The `minmax()` function takes in two parameters - *min* and *max*. It specifies a size range greater than or equal to *min* and less than or equal to *max*. Both these values can be any length values in `px`, `%`, `rem` or even values like `1fr`, `min-content` or `max-content`.

In the previous example, we want the card to be a minimum of the content width and a maximum of `18rem`. That's why we used `auto` as the value for *min* and `18rem` as the value for *max*.

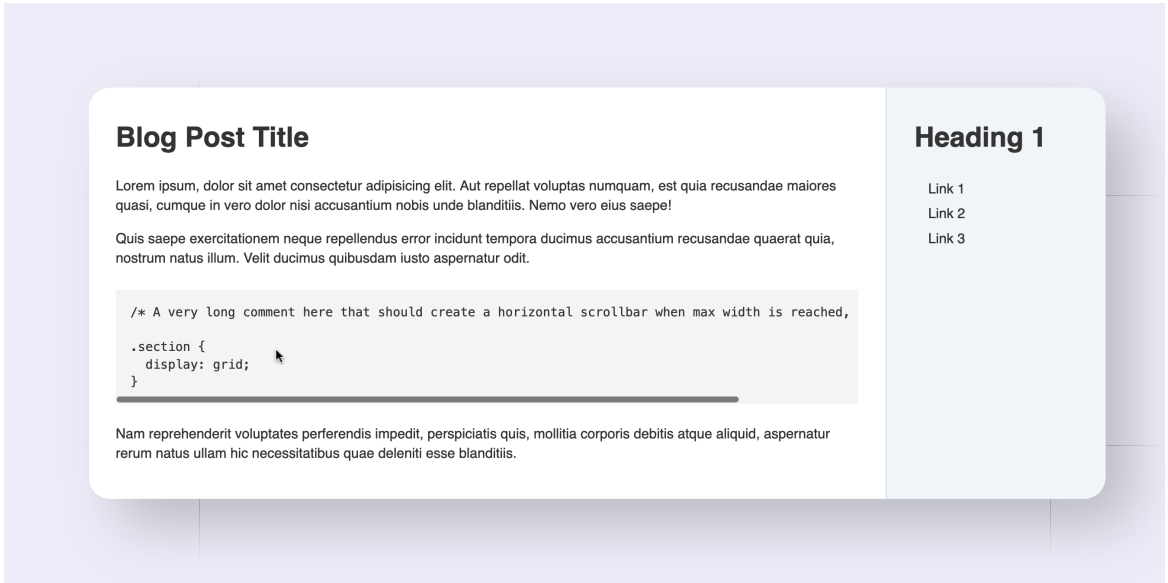
## Blog Post Page with Code Snippet Example 27b

Example Inspired from [CSS Tricks Article - Preventing a Grid Blowout](#)

We looked at creating a simple [page layout with a sidebar](#) using:

```
grid-cols-[22rem,1fr]
```

It's a simple solution and it usually works. But consider this blog post page example with a similar page layout. Here we need to display a code snippet using a `pre` tag. And code snippets can sometimes have long lines of code or comments. When we set `max-width-full` and `overflow-scroll` to the `pre` element we expect it to occupy a maximum of 100% width and display a horizontal scrollbar.



But look at this link, resize the window to a smaller width and see what happens.

▶ Try it out

The main content expands to occupy the full width of the `pre` element making the whole page "blowout". But why?

The `1fr` value stretches the column to occupy remaining space when the content is small, but otherwise, the minimum width is `auto`. So `1fr` is actually equivalent to `minmax(auto, 1fr)`. So, when you add a `pre` element with a huge width, that column occupies a minimum width of the `pre` element. Let's look at the solution for this problem:

### Solution

```
1 <section class="min-h-screen grid grid-cols-[minmax(0,1fr),16rem]">
2   ...
3 </section>
```

▶ Working Demo

Now that the range is `0` to `1fr`, it works fine. If none of this makes sense, just remember one thing - `minmax(0, 1fr)` is always a better option than `1fr`. Which is why, if you look at the Tailwind classes `grid-cols-*`, their equivalent CSS values are:

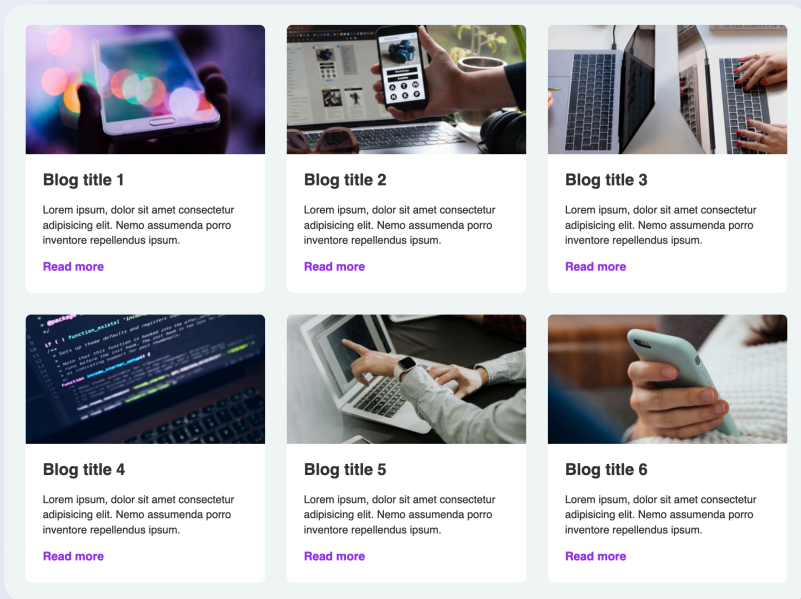
Tailwind Class	CSS Property & Value
<code>grid-cols-1</code>	<code>grid-template-columns: repeat(1, minmax(0, 1fr));</code>
<code>grid-cols-2</code>	<code>grid-template-columns: repeat(2, minmax(0, 1fr));</code>

And so on. But you can simply use the `grid-cols-*` utilities and not worry about any blowout.

## Responsive Grid without Media Queries Example 27c

Remember how we made the [blog\\_posts\\_display](#) responsive by adding media queries at two breakpoints to change the number of columns? Well, we actually don't need to do that. Grid has a way to decide how many columns to create based on the space available. But there's no Tailwind solution for this too. We will be using arbitrary values again.





## HTML

```
1 <div class="container">
2   <div class="item"> .. </div>
3   <!-- Five more item cards -->
4 </div>
```

## Solution

```
1 <div class="container grid grid-cols-[repeat(auto-
  fit,minmax(16rem,1fr))] gap-8">
2   ...
3 </div>
```

If you find these arbitrary values too hard to read, feel free to use custom CSS

```
1 .container {
2   ...
3   grid-template-columns: repeat(auto-fit,minmax(16rem,1fr));
4 }
```

## ▶ Working Demo

Resize your browser window to see that grid is automatically creating more or less columns. Let's break this and understand what's happening and how.

Previously we used `repeat(3, 1fr)` for large screens. Now we have replaced the first value with `auto-fit` and second value with `minmax(16rem, 1fr)`.

Now you know what `minmax(16rem, 1fr)` does. It occupies a minimum of `16rem` width no matter what. And if more space is available, it stretches to occupy more width. But what is `auto-fit`?

## Understanding `auto-fit` Concept

The keyword `auto-fit` tells the browser to handle the number of columns and their sizes such that the elements will wrap when the width is not large enough to fit them in without any overflow. The `1fr` in the second value of `repeat` ensures that in case there's more space available, but not enough to accommodate another full column, that space will be distributed among the other columns, making sure we aren't left with any empty space at the end of the row.

### How the browser calculates

To understand the above example better, assume we have a screen width of `40rem`. The container has a padding of `2rem` on left and right. So, now we're left with

$$40\text{rem} - 4\text{rem} = 36\text{rem}$$

This can easily fit one column of `16rem`. After one column is placed, we are left with

$$36\text{rem} - 16\text{rem} = 20\text{rem}$$

Can we fit another column of `16rem` along with a `gap` of `2rem`? Yes, we can. So after the second column is placed, we are left with:

$$20\text{rem} - (16\text{rem} + 2\text{rem}) = 2\text{rem}$$

Now we have `2rem` of extra space, so that's divided between the two columns and now each column is `17rem` wide.

On the other hand, if the screen width is `36rem` instead of `40rem`, trace the same steps and you'll see that we cannot fit the second column there. So, after subtracting the container padding, we are left with `32rem` - which is the width of that single column.

But you really need not worry about all the above calculation. Ideally you just need one CSS rule to create a responsive grid layout of equally sized columns.

```
1 grid-template-columns: repeat(auto-fit, minmax(<fixed-width-value>, 1fr));
```

Now consider a scenario where you have just one blog post. How do we prevent it from filling up the entire row?

## Solution

```
1 grid-template-columns: repeat(auto-fill, minmax(16rem, 1fr));
```

### ▶ Working Demo

We have replaced the `auto-fit` keyword with `auto-fill`.

## Understanding `auto-fill` Concept

This is very similar to `auto-fit`. In our previous example where there are more than 5 blog posts, you will not be able to notice any difference at all. Try for yourself. Both the keywords give us the same result. But when there are fewer items and more space to fill in items:

- `auto-fit` distributes the remaining space leaving no empty space in the row
- `auto-fill` creates blank columns of the same size as the items.

If this is not clear, this CSS Tricks article - [auto-fill vs auto-fit](#) explains it really well.

Now use this method to make these examples responsive without using media queries:

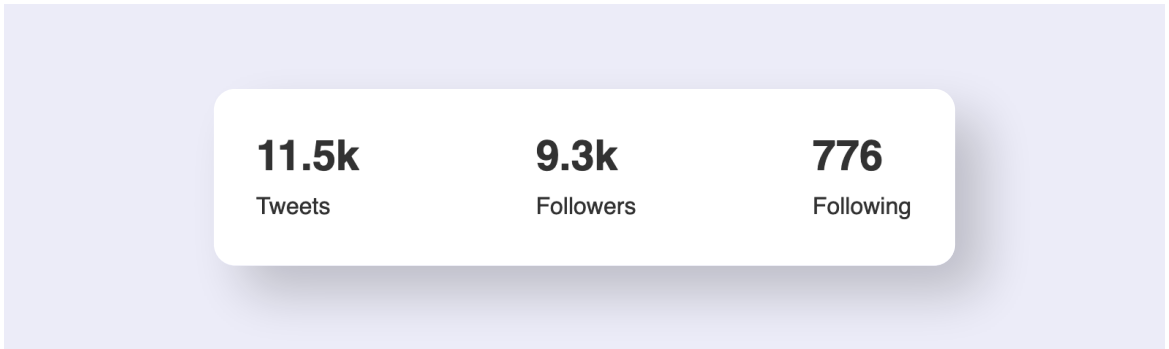
1. [Featured Logos in a Grid](#)
2. [Responsive Pricing Plans](#)

You decide whether to use `auto-fit` or `auto-fill`.

## 28 Grid Auto Flow

### Analytics Section Example 28a

Here is a simple section that shows analytics with numbers and labels. Usually we would create 3 separate `div` elements for this and each div would contain a number and label. But we can avoid those additional divs using grid.



▶ Try it out

#### HTML

```
1 <section>
2   <span>11.5k</span>
3   <p>Tweets</p>
4   <span> ... </span>
5   <p> ... </p>
6   <span> ... </span>
7   <p> ... </p>
8 </section>
```

## Solution

Clearly we need three columns and two rows for this. But when we create a 3 x 2 grid, items start getting placed one by one filling first row and then move to second row. We need to change this default flow, to fill the column first instead of row, by adding the `grid-flow-col` utility class:

```
1 <section class="grid grid-rows-[auto,auto] grid-flow-col justify-between">
2   ...
3 </section>
```

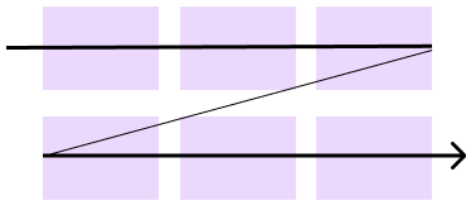
### ▶ Working Demo

In most of the previous examples, we created columns explicitly using `grid-cols-*` and the rows automatically got created based on the content. However, here we are creating two rows using `grid-rows-[auto,auto]`. Columns are automatically created based on the content.

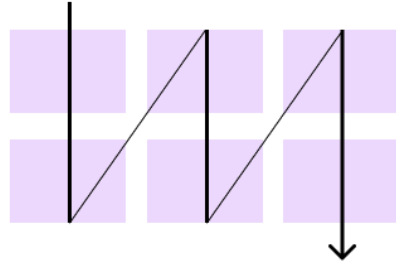
## Understanding Grid Auto Flow Concept

The CSS property `grid-auto-flow` specifies the flow in which the grid items get placed into the rows and columns. By default, the flow is `row`. Which means, items start getting placed one by one filling first row and then keep adding more rows.

The Tailwind equivalent for this property with value `row` is `grid-flow-row` (The default) and for `column` is `grid-flow-col` (Which we used in our previous example).



`grid-auto-flow: row`



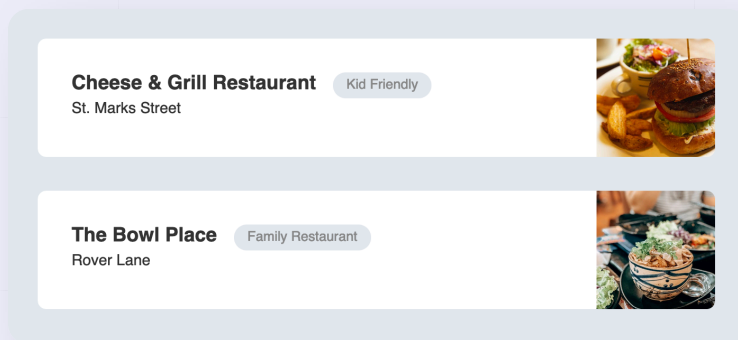
`grid-auto-flow: column`

Tailwind Class	CSS Property & Value	Explanation
<code>grid-flow-row</code>	<code>grid-auto-flow: row;</code>	The items get placed one by one filling one row after the other
<code>grid-flow-col</code>	<code>grid-auto-flow: column;</code>	The items get placed one by one filling one column after the other

## 29 Justify Self & Align Self

### Restaurant Cards with Labels Example 29a

Let's say we need to list restaurants with name, street, label and a picture just like the screenshot below. You already know how this is done with flexbox. Now let's see how to do this using grid.



▶ Try it out

### Markup

```
1 <div class="container grid grid-cols-[auto,auto,1fr]">
2   <div class="info"> ... </div>
3   <span class="label"> ... </span>
4   <img ... >
5 </div>
```



## Solution

```
1 <div class="container grid grid-cols-[auto,auto,1fr]">
2   ...
3   <span class="label self-start"></span>
4   <img class="justify-self-end" ... >
5 </div>
```

### ▶ Working Demo

We have created 3 columns where the width of first two columns is `auto` and the last column is `1fr`. So, the info and label columns occupy as much space as needed by the content and the remaining space is assigned to the image column. We have used `justify-self-end` on the image to push the image to the right.

Also, since the grid items stretch to occupy full height by default, the label stretches full height. To prevent this, we have used `self-start` on the label.

Let's learn more about these two new properties.

## Understanding Justify Self and Align Self Concept

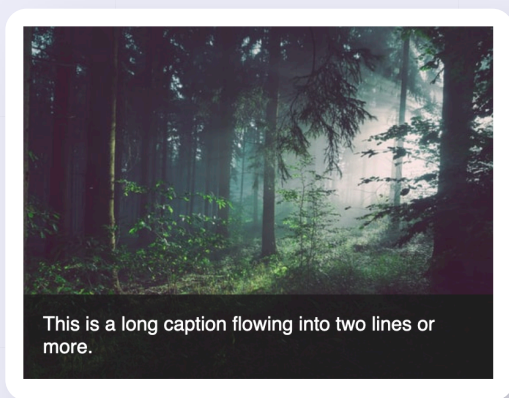
The utilities `justify-self-*` is used on a **grid item**. When the content of the item is smaller than the width of the column, we can use this property to control the alignment along the *row axis* (horizontal direction).

The utilities `self-*` is also used on a grid item. When the content of the item is shorter than the height of the row, we can use this property to control the alignment along the *block axis* (vertical direction).

The available utilities are similar to that of `justify-items-*` and `items-*`.

## Caption at the Bottom of Image Example 29b

Here's an example where you wish to place a caption with a transparent overlay on the image sticking to the bottom. Usually this is done with absolute positioning, but there's one problem there. When the image is too small (on mobile screen) and the caption cannot fit in the dimensions, a part of the text gets hidden. But if we implement this with grid, the image expands to fit the content within.



▶ Try it out

### Markup

```
1 <figure>
2   <img ... >
3   <figcaption> ... </figcaption>
4 </figure>
```

## Solution

```
1 <figure class="grid">
2   <img class="col-start-1 col-end-2 row-start-1 row-end-2" ... >
3   <figcaption class="col-start-1 col-end-2 row-start-1 row-end-2 self-
  end">....</figcaption>
4 </figure>
```

### ▶ Working Demo

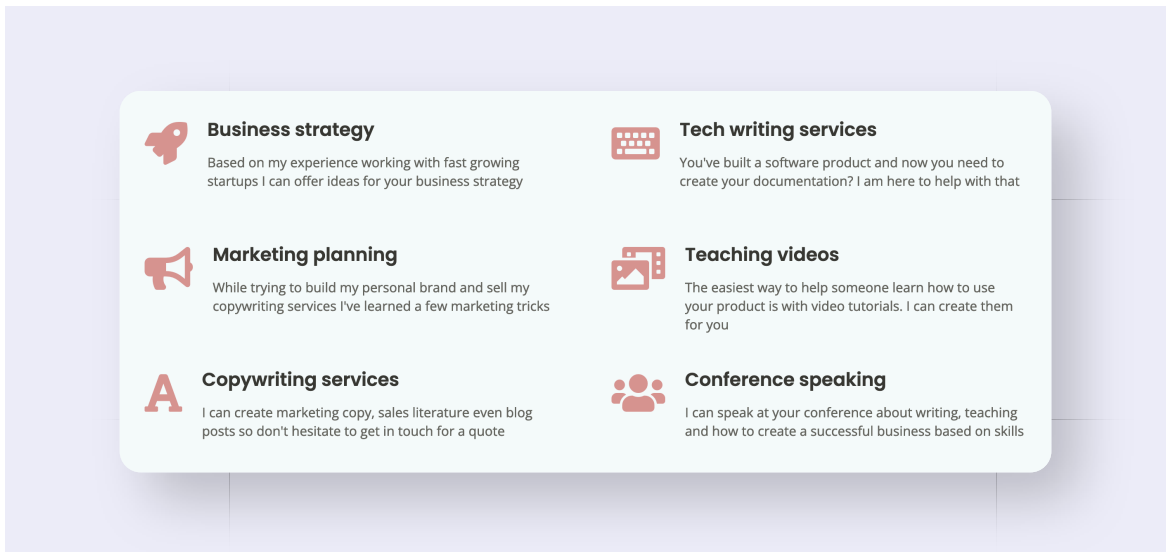
We are trying to create only one grid cells and fit both the items into the same cell using `col-start-1`, `col-end-2`, `row-start-1` and `row-end-2`. That makes the items overlap! Yes, it's possible to create overlapping elements with grid. So then, we use `self-end` to push the caption down to the bottom.

# Comprehensive Examples for Grid & Flexbox

## Services Section Example 30a

Example from [Inovatik Template](#)

This is a responsive services section in a grid format from a template by Inovatik. On mobile screens, two columns collapse into one. This a great example of flexbox within a grid layout.



The whole section is a grid with two columns and three rows on desktop. The flow of the grid is column. And each service is a flex container with the icon and text being flex items.

▶ Try it out

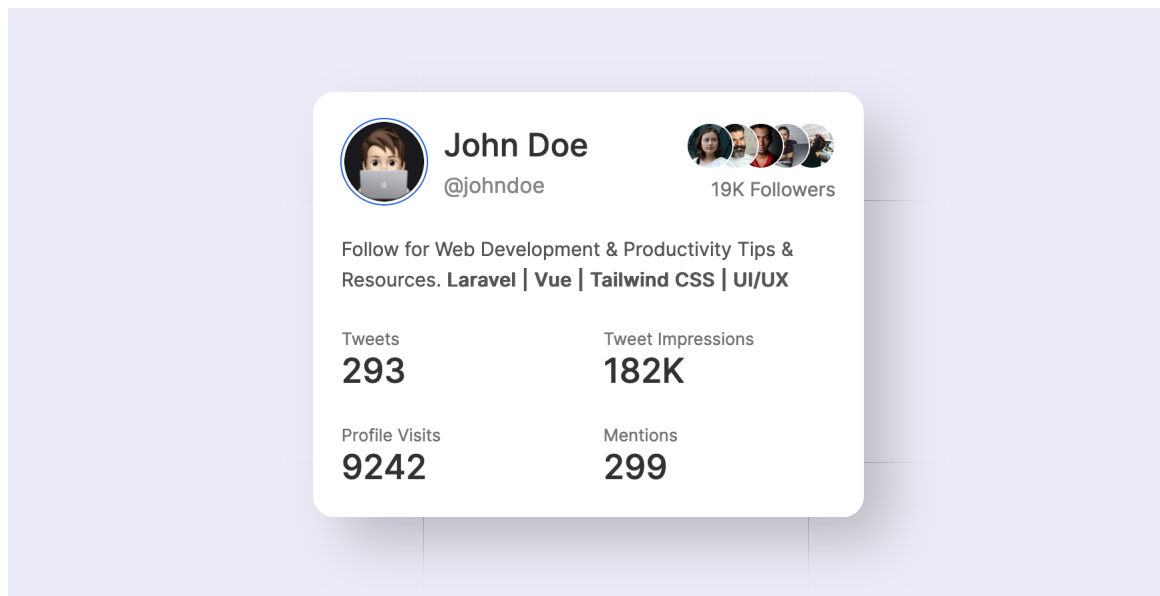
**HINT :** Use the utilities `grid-rows-*` and `grid-flow-col` for the grid. And make each grid item a flex container.

▶ Working Demo

## Twitter Monthly Summary Card Example 30b

Example contributed by [Naresh](#)

Look at this card with one month summary of a Twitter profile along with some profile info. This is a good example of flexbox and grid together in a component.



The header part is best implemented with a flex container, although you can choose to use grid even for that. The statistics at the end is a simple grid layout with two columns and two rows.

▶ Try it out

**HINT :** Use `items-*` and auto margins within flexbox. Use `row-reverse` direction for the followers' images. For the grid, simply use `grid-col-*` and `gap`.

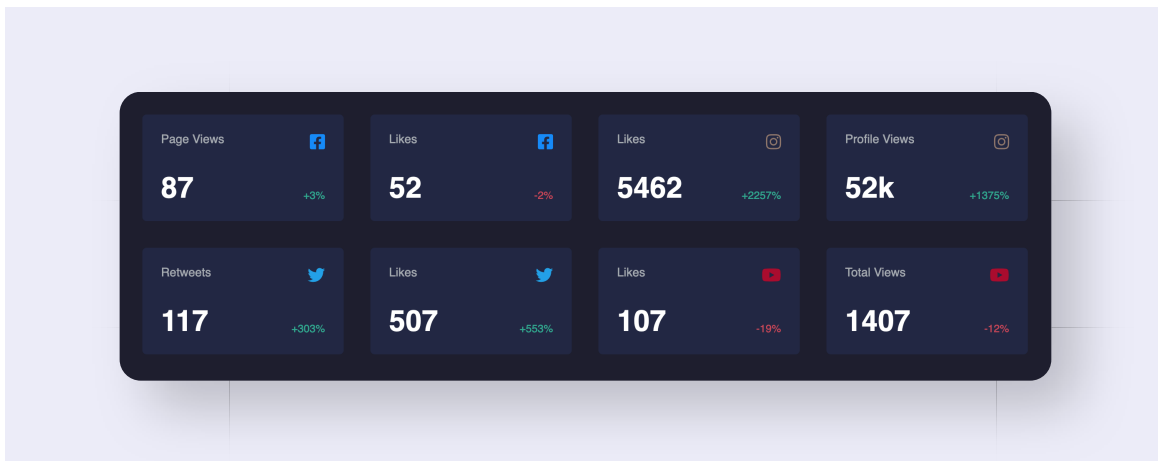
▶ Working Demo

I haven't made this responsive for smaller screens. You can try it out on your own.

## Social Media Dashboard Example 30c

Part of a challenge from [Frontend Mentor](#)

This example is part of the social media dashboard challenge. It's a brilliant example for grid within grid.



Use a grid layout for the entire dashboard. And then, make each grid item also a grid container.

▶ Try it out

**HINT:** Use the utilities `grid-cols-*`, `justify-*`, `content-*`, `justify-self-*`, `self-*` and `gap`.

▶ Working Demo

Check the CSS tab under `/* Important */` comments for the solution. I haven't made this responsive for smaller screens. You can try it out on your own.

## Conclusion

Congratulations! 🎉 You have reached the end of this book. This is a lot of content you have consumed. I hope you took enough time to practise each of the examples, gave enough thought to why one approach is better than other and tried to understand every concept looking at its application.

Do let me know how this book helped you by sending a mail to [contact@shrutibalasa.com](mailto:contact@shrutibalasa.com). And watch out for updates to the book. Thank you 🙏