CSS Flex & Grid

Complete Guide with Real World Examples and Code Snippets

Vanilla 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 - Vanilla CSS

Version 1.0

Published Online: November 5, 2021

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

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

Prerequisites

Throughout this book I will assume that you know the basics of CSS. You need not be good at it, but just need to know about properties like `width`, `height`, `margin`, `padding`, `font`, `color`, `background`, `border`, `position`, `float` and concepts of viewport, responsive web design and media queries.

What not to expect

1. Do not expect to see the concepts explained in the same order in which they are usually covered in other tutorials.
2. 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.
3. Some CSS properties are supported by latest versions of major browsers, but still lack support in older versions of a few browsers as of writing this eBook. I will not be talking about browser support for each of the properties. I recommend cross-checking with the Can I use tool for browser support before implementing flex & grid in your projects.
4. These two concepts are **really** huge! There are a few topics that I don't cover in this book, which you might encounter in other courses. Few reasons for the same:
    1. The browser support is very low for these topics (Example: subgrid) or
    2. There's hardly any real world use case for those topics.
    3. They're too complex to cover in the same book.

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 CodePen 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 Flexbox or Grid. 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 CSS code snippet (usually just a few lines) you can add to the above CodePen link. Then it works! Even without knowing the concept, just looking at the CSS code, 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 try and understand the concept and the CSS property we just used, and also look at other values that can be used with this property. This is labelled **Concept**

STEP 5 : You might see some more examples next, to practice the concept that you just learned with different values. Each of these examples have solutions and 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 these CSS properties 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 next to reinforce the knowledge you already have.

CodePen Links

1. These are private *Pens* available only to those with the links. Kindly do not share these links individually anywhere else.
2. In the CSS tab, you will first find all the generic styles like `color`, `margin`, `padding`, `font` etc., within the comments block `/* Generic Styles */`. You can usually ignore the styles within this block because these have nothing to do with flexbox or grid. So, every time you open any CodePen link here, scroll to end of the CSS panel and focus only on the code after `/* Generic Styles End */` comment. That's where the most important code lies.

Reach Out

Feel free to send a mail to contact@shrutibalasa.com or send a Direct Message on Twitter - @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.

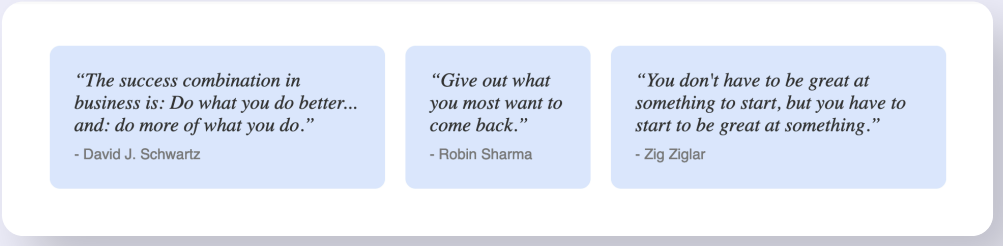
Flexbox

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 CodePen link for you to try achieving this layout using any of the CSS properties 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** CSS rule? Let's see how.

HTML

```
<div class="container">
  <div class="quote">...</div>
  <div class="quote">...</div>
  <div class="quote">...</div>
</div>
```

Solution

Now here's the **CSS** rule:

```
/* Other styles here */
.container {
  display: flex;
}
```

Here's the full working demo. Tada! 💥

▸ Working Demo

Resize the output panel, rearrange the quotes or add longer ones. Notice how ***flex***ible 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, `.container` is the flex container, while `.quote` elements are the flex items. And as you just saw, adding **`display: flex`** 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.

Once you have a flex container and some flex items, there are multiple other CSS properties that can be added to these elements to control the dimensions, alignment, spacing and more. We will be looking at all those properties 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.

HTML

```
<div class="wrapper">
  <div class="menu">
    <a class="active" href="#">Profile</a>
    <a href="#">Notifications</a>
    <a href="#">Payments</a>
    <a href="#">Settings</a>
  </div>
</div>
```

Solution

Now add these two properties in **CSS** to the `.menu` selector

```
.menu {
  /* Other styles here */
  display: flex;
  justify-content: space-between;
}
```

There you go!

▸ Working Demo

Apart from `display: flex`, we added just one more rule - `justify-content: space-between`. Let's learn more about this property.

Understanding `justify-content` Concept

Before we understand this property, there's something else you need to know. The moment we add `display: flex` 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 CSS property **`justify-content`** is used to control spacing of the flex items in the direction they are placed. In our above example, it's the horizontal direction. `space-between` is one of the values we just used.

Following values can be given to `justify-content`:

`flex-start` *(Default value)*

All items are placed at the beginning of the container with no spaces

`flex-end`

All items are placed at the end of the container with no spaces

`center`

All items are placed at the center with no spaces

`space-between`

All items are spaced out as much as possible with first item at the beginning and last item at the end

`space-around`

Space *before* the flex items and *after* the flex items are half as much as space between the items

`space-evenly`

Space before, after and between the items are same

You can see the difference between these values below:

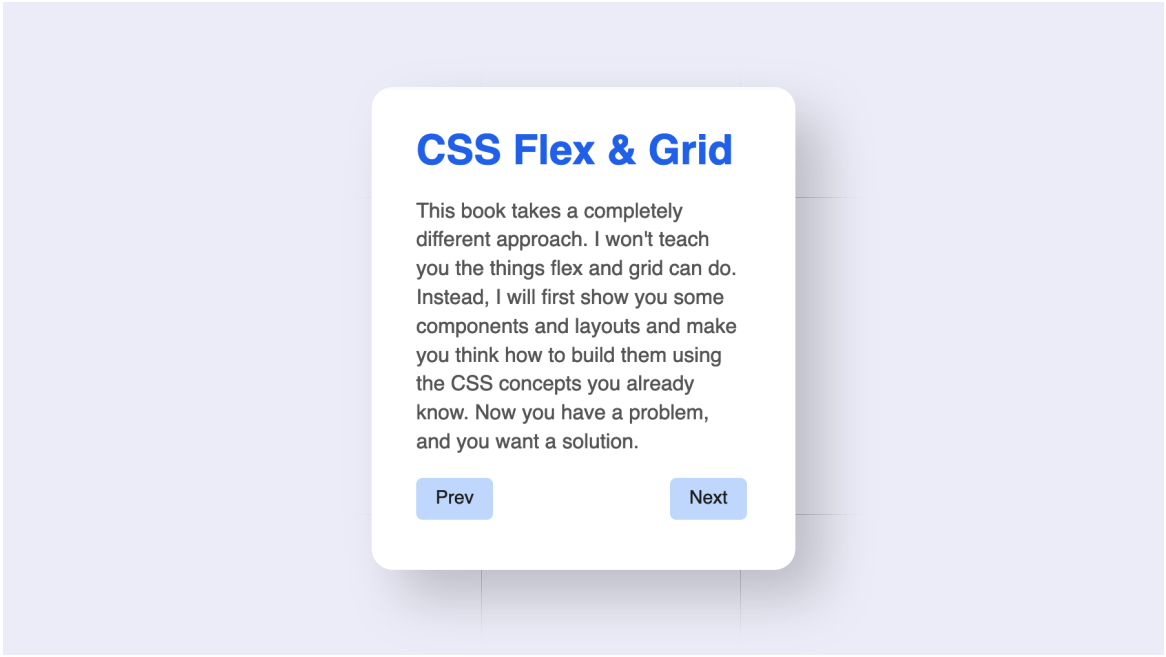
Open the working demo, resize the output panel and see how the items move.

▸ Working Demo

Let's look at some more examples where this property 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-content` property used for alignment.

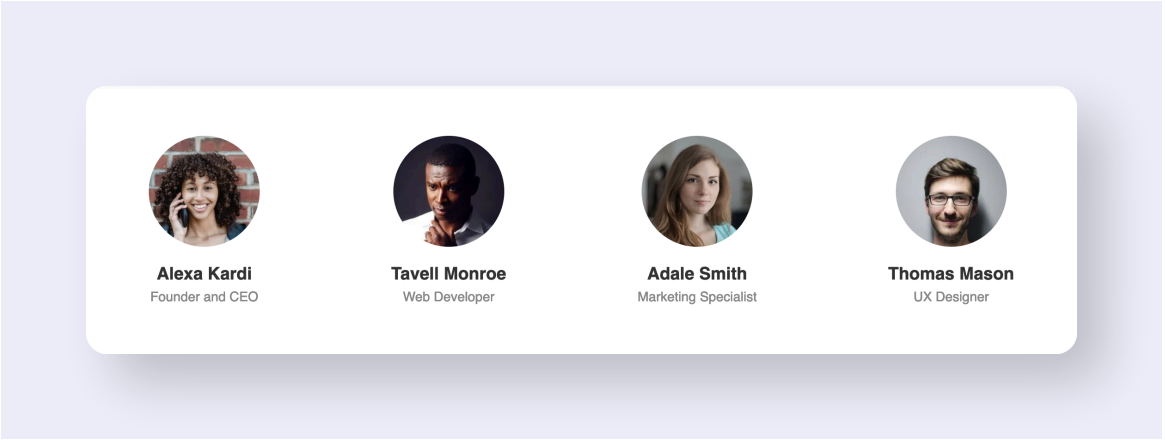


Now that you have seen one example, try this out on your own and cross check with the working demo.

▸ 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-content` property set to `space-around` for the container.



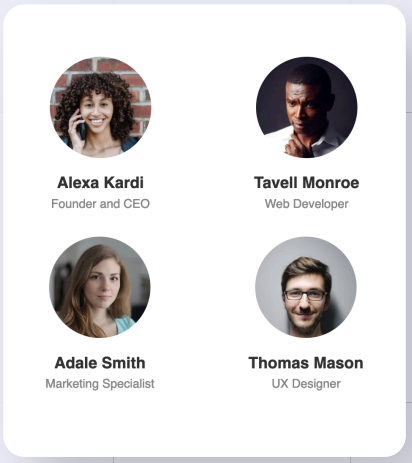
Try it out yourself in the CodePen link below. HINT: Add styles to the `.container` selector at the end of the CSS code.

▸ 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 this rule to the flex container:

```
.container {
  flex-wrap: wrap;
}
```

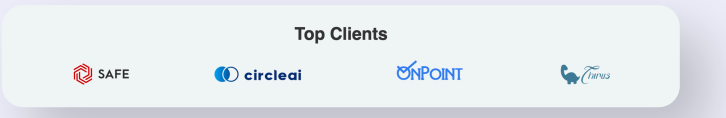
▸ Working Demo

Understanding `flex-wrap` Concept

The `flex-wrap` property decides whether to wrap the flex items if you run out of space or not. The default value is `nowrap` which is why the child items do not move into the next row automatically. When you set the value to `wrap`, the above responsive behaviour can be achieved. 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-content: space-around` for the spacing and the `flex-wrap` property to wrap the logos.



First three logos contributed by Gokul

▸ Try it out

HTML

```
<div class="logos">
  <img ...>
  <img ...>
  <img ...>
  <img ...>
</div>
```

Solution

```
.logos {
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
}
```

▸ Working Demo

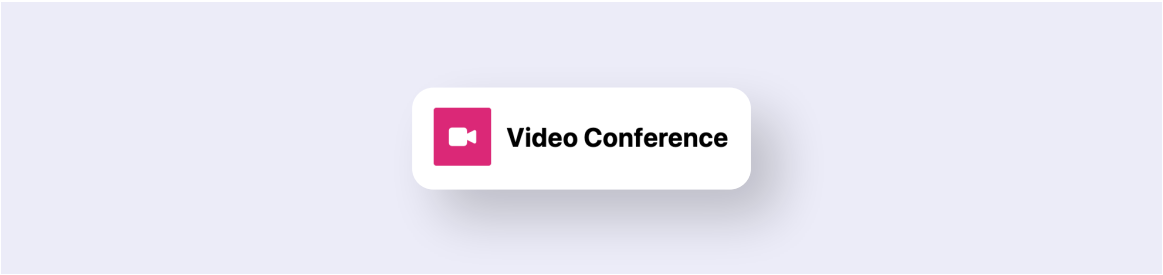
This property also has another value `wrap-reverse`. Try that out to see the difference.

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?

HTML

```
<div class="icon-wrap">
  <span class="icon material-icons">videocam</span>
  <span class="icon-text">Video Conference</span>
</div>
```

▸ Try it out

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

Solution

Instead of the `vertical-align` properties, add these two rules to the `.icon-wrap` selector.

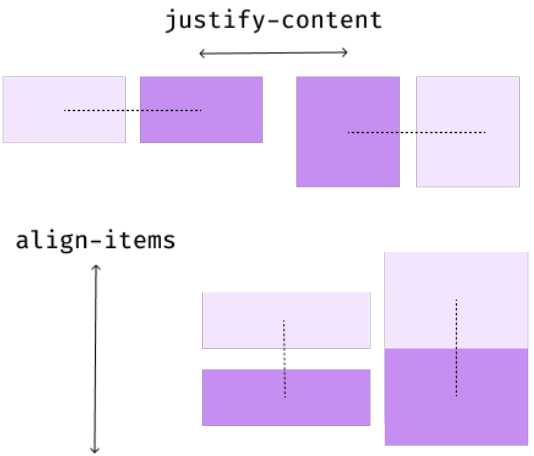
```css
.icon-wrap {
  /* Existing styles here */
  display: flex;
  align-items: center;
}
```

▸ Working Demo

Apart from `display: flex`, we added just one more rule - `align-items: center`. Let's learn more about this property.

Understanding `align-items` Concept

The `justify-content` property allows us to control the spacing and alignment of the flex items in the direction they are placed (Horizontally in all our previous examples). While `align-items` property allows 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-content` can be used to align the items horizontally, and `align-items` can be used to align items vertically. This is useful especially when the height of each item is different.

Following values can be given to `align-items`:

`stretch` *(Default value)*

All items are stretched to fill the container

`center`

All items are aligned to the center of the container

`flex-start`

All items are aligned to the beginning of the container *(at the top in case of the above example)*

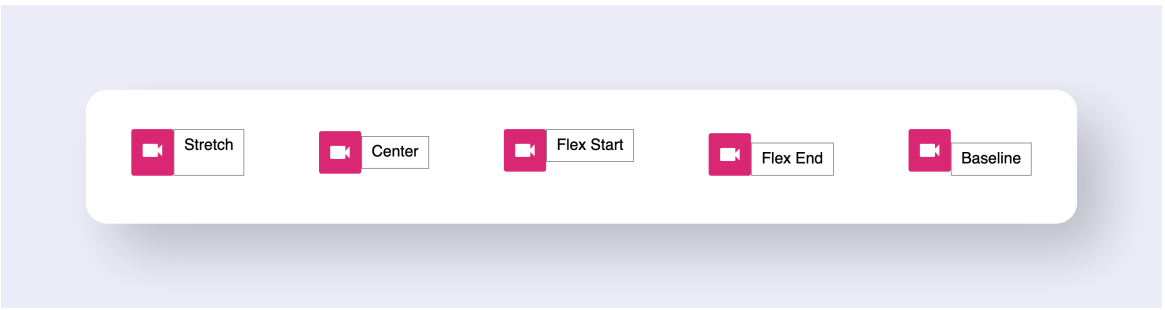
`flex-end`

All items are aligned to the end of the container *(at the bottom in case of the above example)*

`baseline`

All items are positioned such that the base aligns to the end of the container *(will we talk about this soon)*

You can see the difference between these values below:



▸ Working Demo

To understand the effect of `baseline` value, replace the icon with an alphabet by changing

```
1  <span class="icon material-icons">videocam</span>
```

to

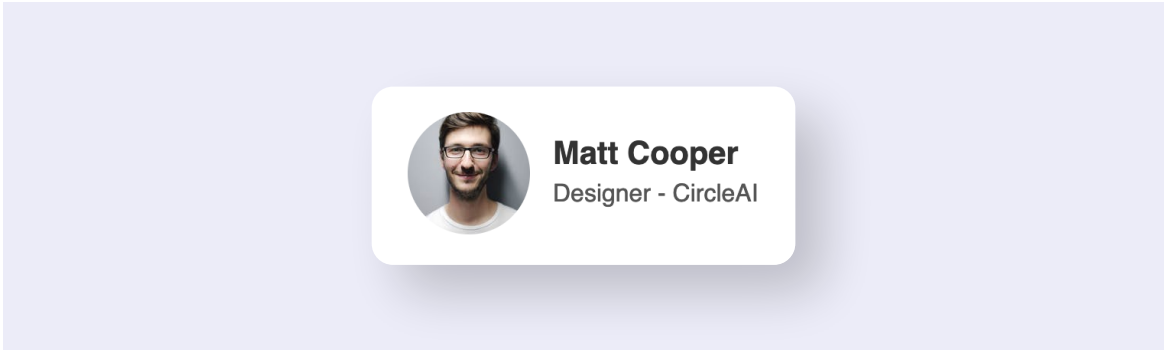
```
1  <span class="icon material-icons">V</span>
```

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 values are `stretch` and `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 `align-items` property with value `center` is very useful for such requirements:

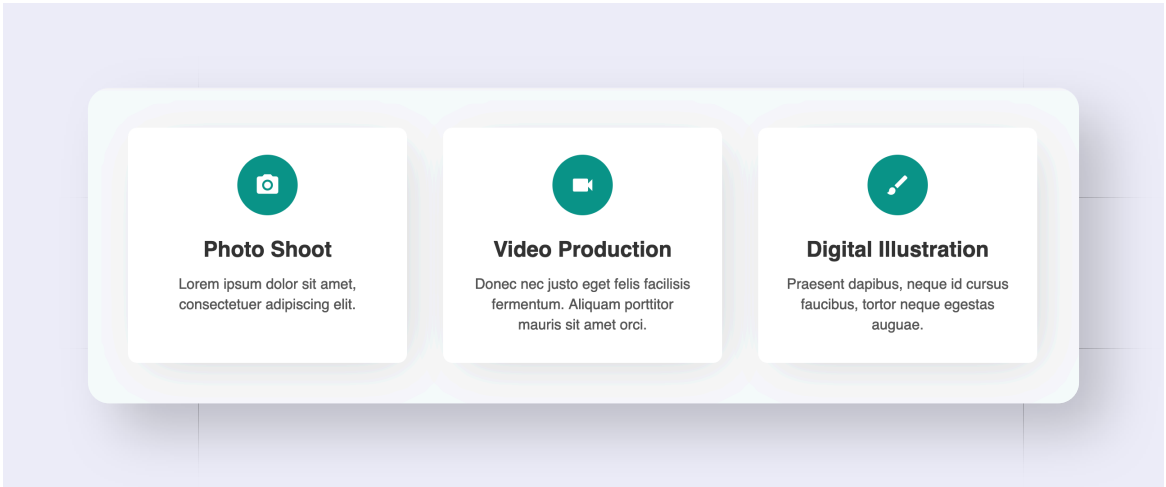


Try doing this yourself before looking at the working code

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 value `stretch` of `align-items` property.



Working Demo

To understand the difference better, change the rule:

```
align-items: stretch;
```

to

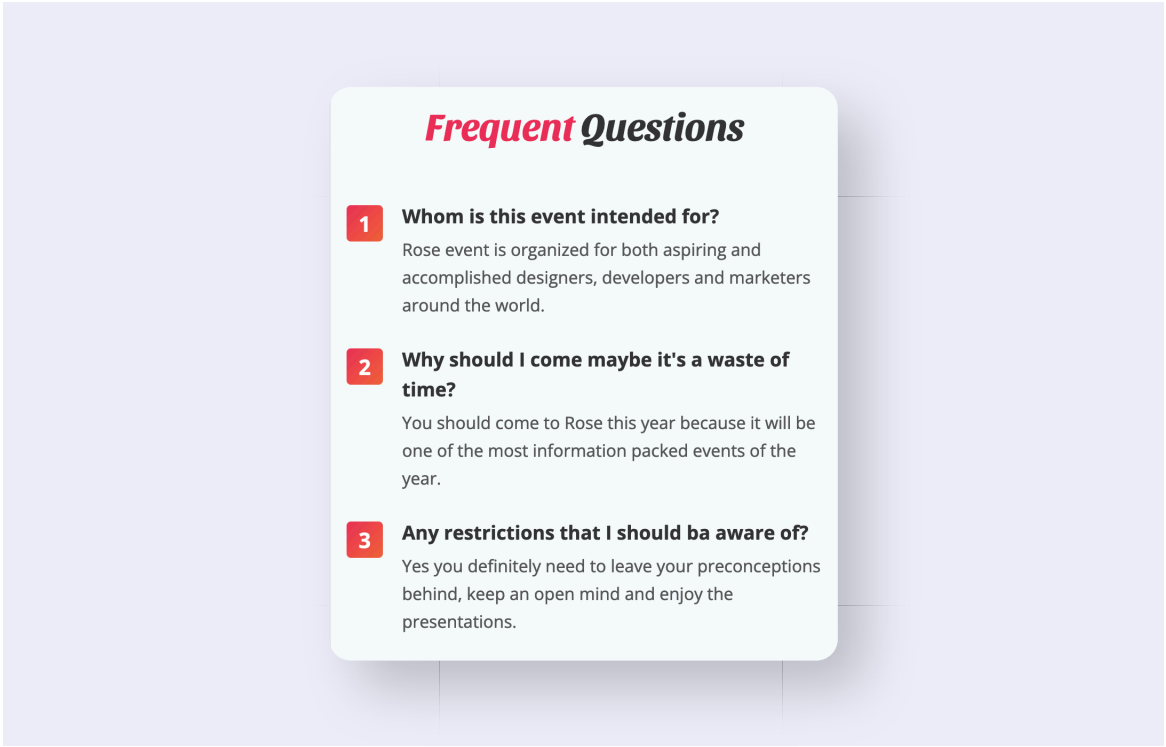
```
align-items: flex-start; /* Or flex-end */
```

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.

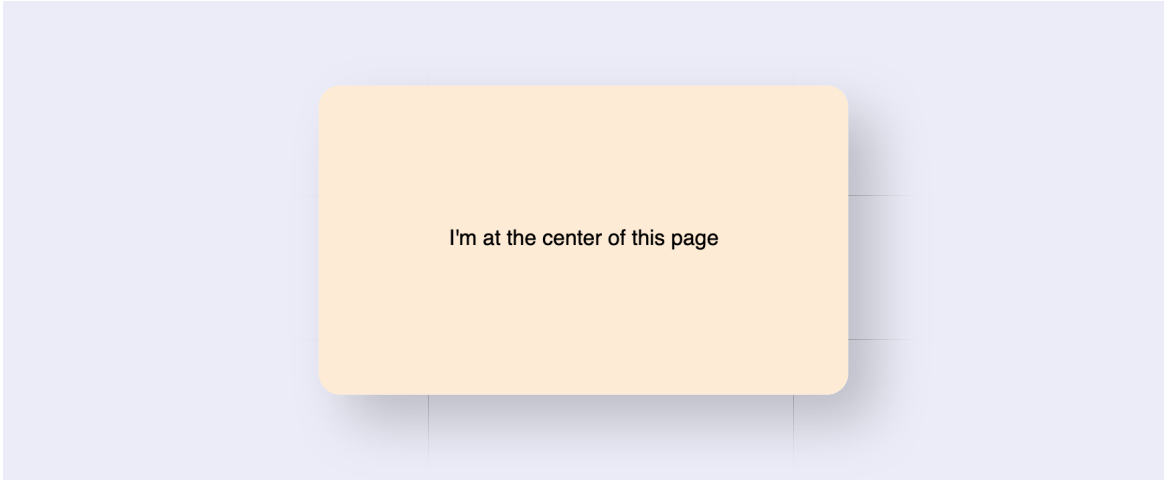


Try using the `align-items` property to make this happen before looking at the working demo.

▸ 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 properties `justify-content` and `align-items`, it's super easy.



HTML

```
<div class="container">
  <div class="item">
    ...
  </div>
</div>
```

Solution 1

```
.container {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

In all the previous examples, we always used more than one flex item. But here, you need to have only one flex item that you wish to center. Below is the working demo where the `.container` takes up full width using `width: 100%` and full page height using `height: 100vh`

▸ Working Demo

Try changing the width and height above to see how the `.item` still remains centered within the `.container`.

Solution 2

There's another way you could achieve the same result

```
.container {
  display: flex;
  justify-content: center;
}
.item {
  margin: auto;
}
```

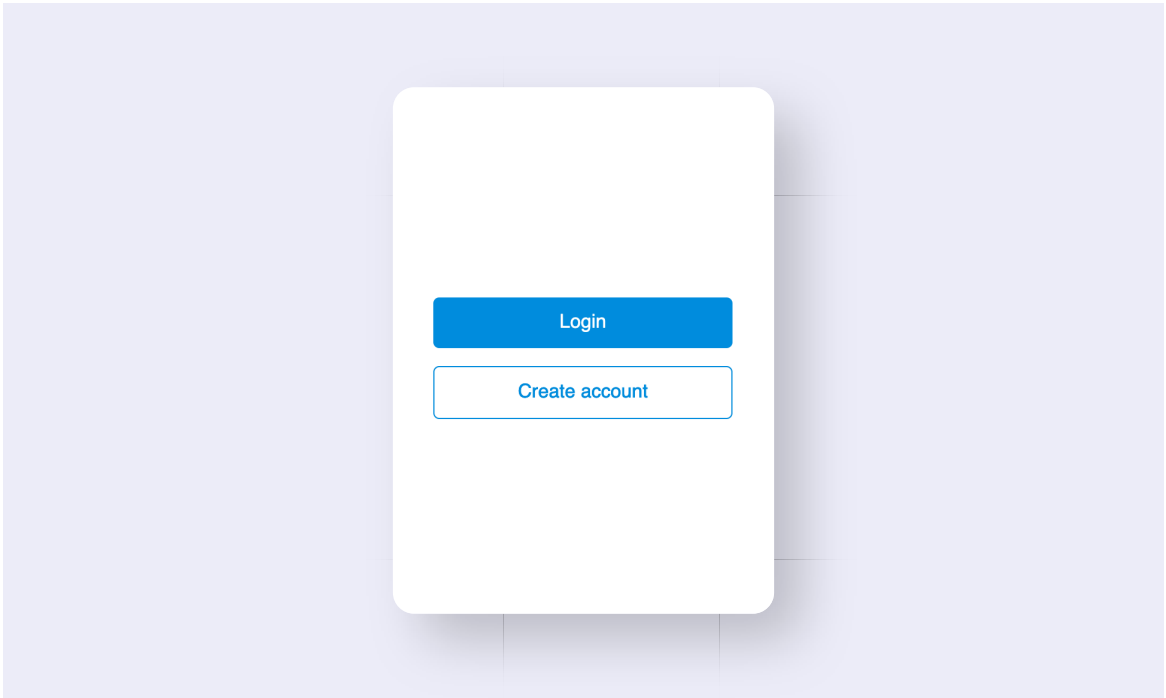
▸ Working Demo

Here, instead of using `align-items` on container, we have used `margin` property on the flex item. You can use any of the two methods 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

HTML

```
<div class="wrapper">
  <div class="container">
    <a href="#" class="link login-link">Login</a>
    <a href="#"class="link signup-link">Create account</a>
  </div>
</div>
```

There are multiple approaches to this. In case you used flex, you might have tried this:

Possible Solution

```
.wrapper {
  display: flex;
  align-items: center;
}
.container {
  width: 100%;
}
```

While the above solution works, there's a better approach. You can instead try this:

Better Solution

```
.container {
  height: 100%;
  display: flex;
  flex-direction: column;
  justify-content: center;
}
```

▸ 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 `display: flex` 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-direction` property with `column` value.

This property has the following different values:

`row` *(Default value)*

All items are placed in a single row from left to right

`column`

All items are placed in a single column from top to bottom

`row-reverse`

All items are placed in a single row from ***right to left***

`column-reverse`

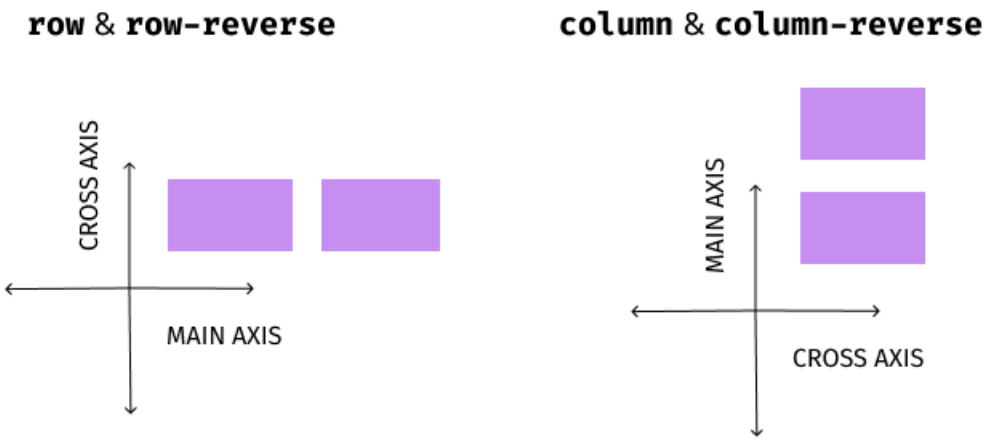
All items are placed in a single column from ***bottom to top***

At first, it appears that using *flexbox* with `flex-direction: column` 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-direction: column` that we will explore further.

Before that, I want you to notice one thing. In the above example, we used `justify-content: center` to center the items vertically. In Example 4a and Example 4b , we used `align-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.

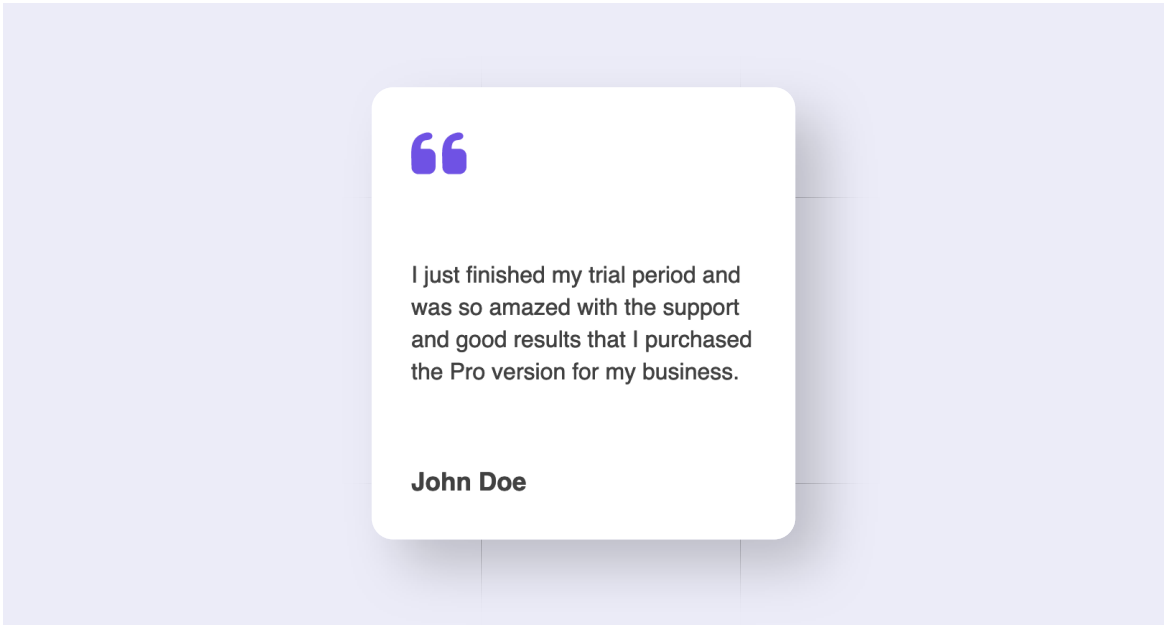


The property `justify-content` controls alignment along the **main axis**, while the property `align-items` controls alignment along the **cross axis**. In Example 5a, our flex direction is `column`. So vertically centering needs alignment along its main axis. That's why we need to use `justify-content`.

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 direction `column` and see if you can get the desired result

▸ Try it out

Solution

We have applied the following 4 CSS rules to the container `.card`

```
.card {
  /* Other styles here */
  display: flex;
  flex-direction: column;
  align-items: flex-start;
  justify-content: space-between;
}
```

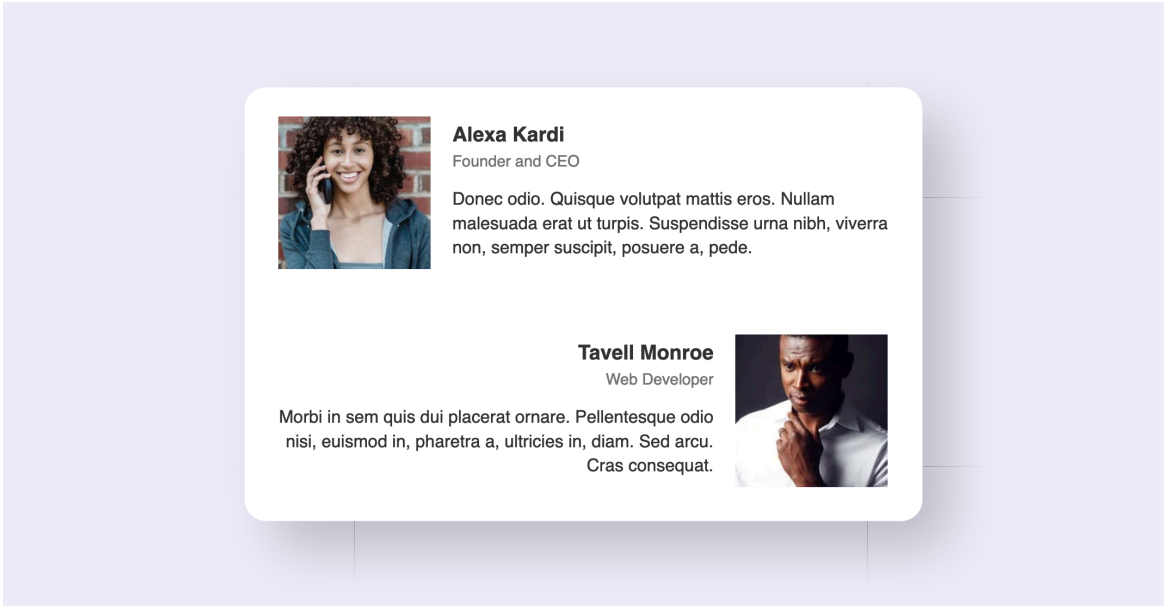
▸ Working Demo

By default, the value of `align-items` is `stretch`. So without the rule on line 5, the icon image stretches full width. The rule `justify-content: space-between` is what makes the child items space out 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.

HTML

```
<div class="profile">
  <img ... >
  <div class="profile-description">...</div>
</div>
<div class="profile">
  <!-- Reverse the order -->
  <div class="profile-description">...</div>
  <img ... >
</div>
```

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

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

▸ Try it out

Solution

```
.profile:nth-child(even) {
  flex-direction: row-reverse;
  text-align: right;
}
```

▸ Working Demo

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 flexbox property for this)*

▸ Try it out

HTML

```
<div class="container">
  <input id="#email" ... >
  <button ... >Subscribe</button>
</div>
```

We already have `display: flex` on `.container`. Next we need to add the below CSS rule to the `#email` element.

Solution

```
#email {
  flex-grow: 1;
}
```

▸ Working Demo

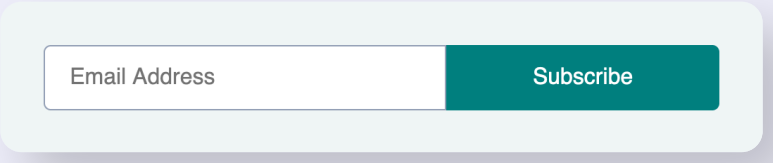
This just works! Resize the browser and it's fully responsive.

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

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 `flex-grow` is 0. By setting the value of `flex-grow` property to a number greater than 0, which 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: 1` 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.

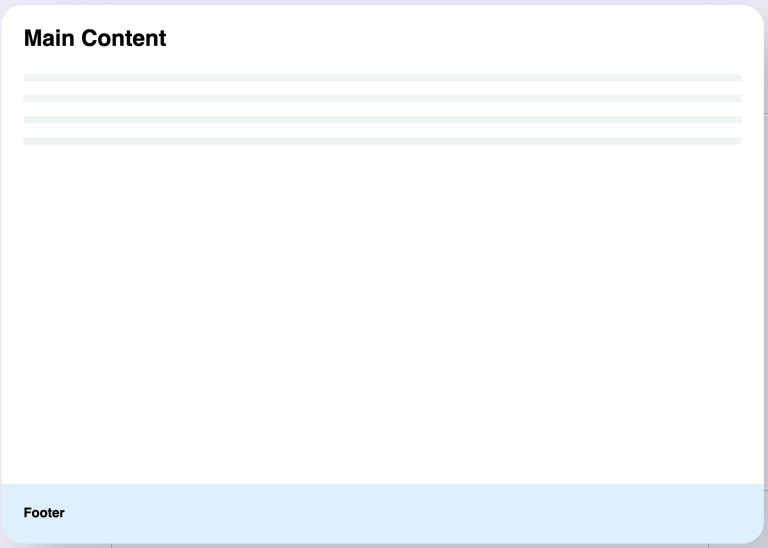
When `flex-grow: 1` is added to two flex items, the left over space is divided into two parts and added to the ***initial*** widths of those two items. Since the text input's initial width was more than that of the button, it occupies more space. It doesn't end here. What if we add `flex-grow: 2` to 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 property
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

HTML

```
<div class="container">
  <div class="main">...</div>
  <footer>...</footer>
</div>
```

We need to first add a `min-height: 100vh` to the `.container`. Otherwise nothing will work. Then make it a flex container with `flex-direction` set to `column`. At last, set `flex-grow` property of the `.main` element to 1.

Solution

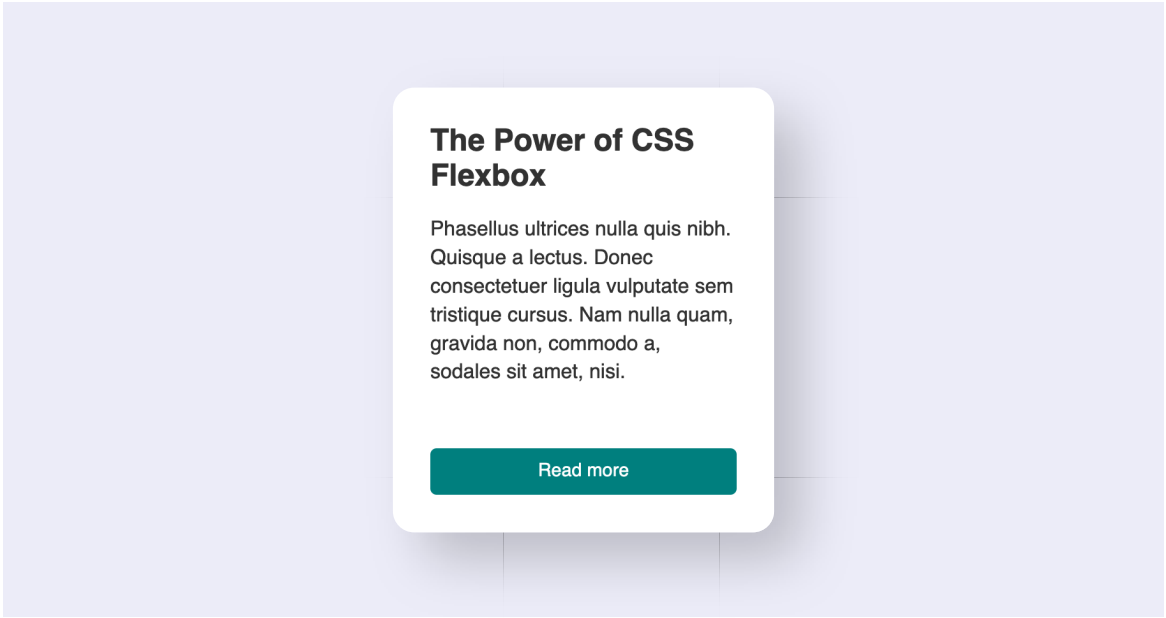
```
.container {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}
.main {
  flex-grow: 1;
}
```

▸ 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.

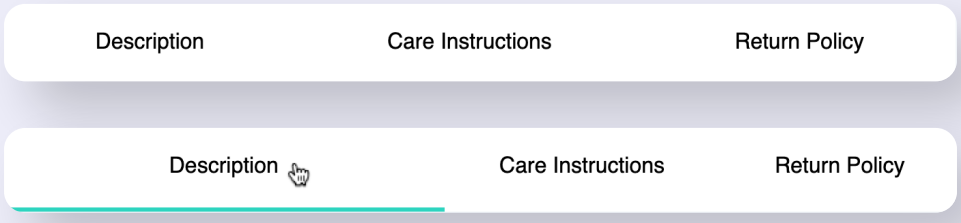


Since this is very similar to the previous example, I would encourage you to try it out first before looking at the working demo.

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.



Can you try and achieve this by changing the `flex-grow` value on hover?

▸ Try it out

HTML

```
<ul>
  <li>...</li>
  <li>...</li>
  <li>...</li>
</ul>
```

Solution

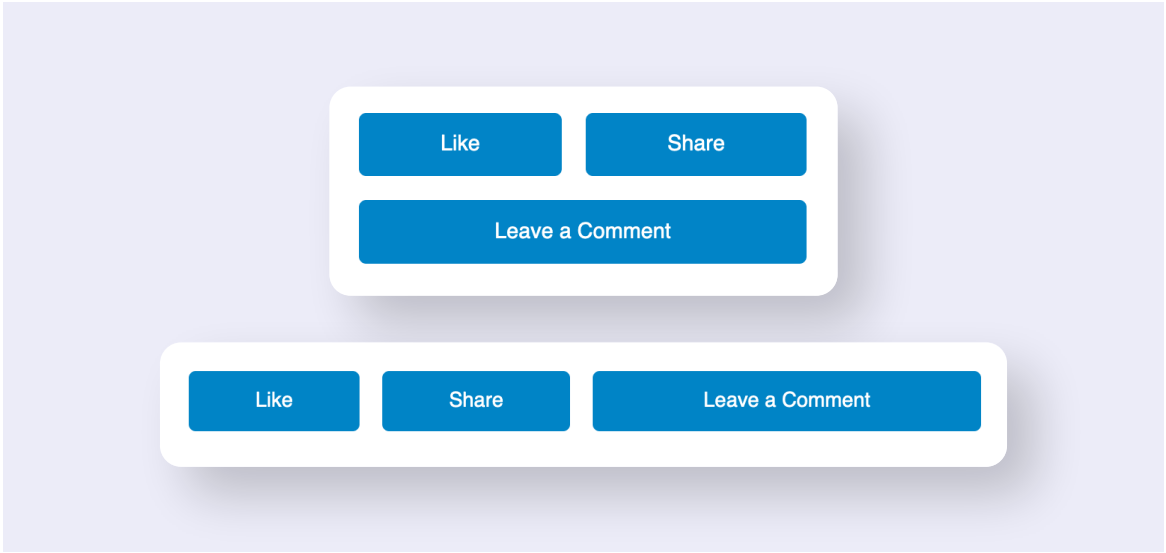
Initially, all the tabs are set to `flex-grow: 1` and on hover, we increase the value of `flex-grow` to any number depending on how wide you want the active tab to be.

```
ul {
  display: flex;
}
ul li {
  flex-grow: 1;
}
ul li:hover {
  flex-grow: 3;
}
```

▸ 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

HTML

```
<div class="container">
  <button type="button"> ... </button>
  <button type="button"> ... </button>
  <button type="button" id="comment"> ... </button>
</div>
```

Solution

```
.container {
  display: flex;
  flex-wrap: wrap;
}
button {
  flex-grow: 1;
}
#comment {
  flex-grow: 2;
}
```

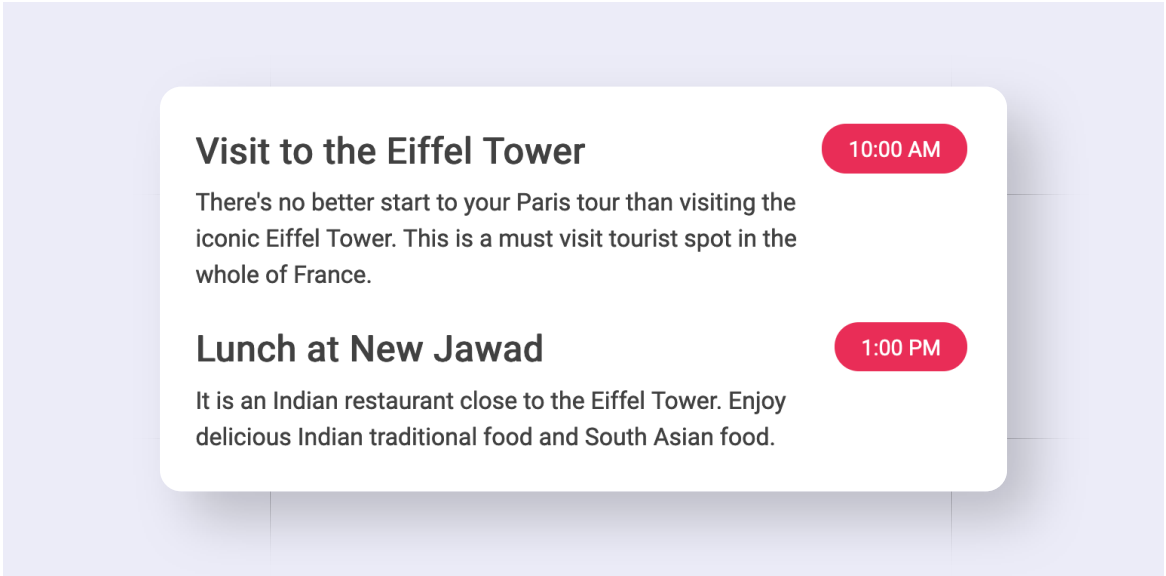
▸ Working Demo

We will look at more examples related to `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 10:00 AM

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

▸ Try it out

Let's see how to get this working using another **flex item's** property `flex-shrink`

HTML

```
<div class="container">
  <div>
    ...
  </div>
  <span>10:00 AM</span>
</div>
```

Solution

```
span {
  flex-shrink: 0;
}
```

This will prevent the element from *shrinking*.

▸ Working Demo

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

Understanding `flex-shrink` Concept

The default behaviour of flex items is to shrink to fit in the width of a single row / height of the 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 `div` shrinks more than the smaller one by default. This is because, the default value of `flex-shrink` for each flex item is **1**.

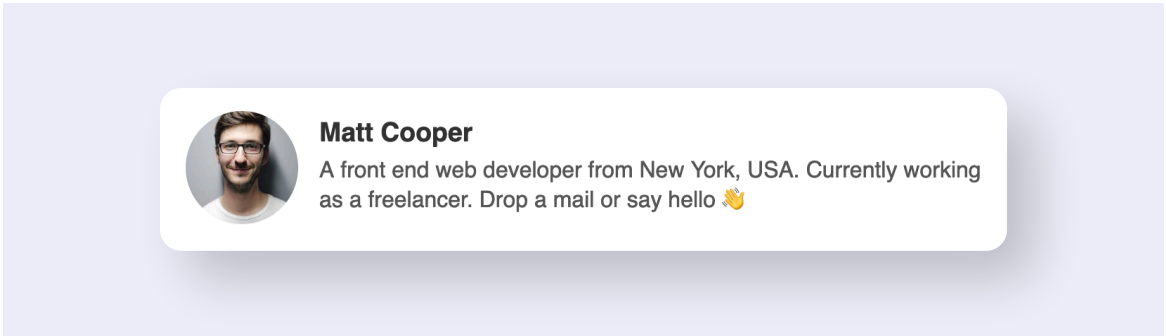
In our previous example, we changed this value to **0**, hence preventing the item from shrinking. The other item shrinks to fit. The way **grow factor** specifies how much additional space the item should occupy, the **shrink factor** specifies how much space should be reduced from the flex item's initial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

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.



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`

HTML

```
<div class="profile">
  <img src=" ... " class="profile-img">
  <div class="profile-info">
    ...
  </div>
</div>
```

Solution

```
.profile-img {
  flex-shrink: 0;
}
```

▸ **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. Split the screen horizontally on large screens. Add a breakpoint of **767px**, below which the screen splits vertically. (Of course, you can use the mobile first approach if you wish)

▸ Try it out

HTML

```
<div class="container">
  <div class="split"></div>
  <div class="split"></div>
</div>
```

If you were able to solve this, you must have added a `width: 50%` to the flex items `.split` on large screens. And on small screens, `height: 50%` and resetting the `width` to `100%`. Or, you might have added a `flex-grow: 1` to both the flex items.

Possible Solution 1

```
.split {
  width: 50%;
}
@media screen and (max-width: 767px) {
  .split {
    width: 100%;
    height: 50%;
  }
}
```

Possible Solution 2

```
.split {
  flex-grow: 1;
}
```

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

```
.split {
  flex-basis: 50%;
}
```

▸ Working Demo

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

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 - size being 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

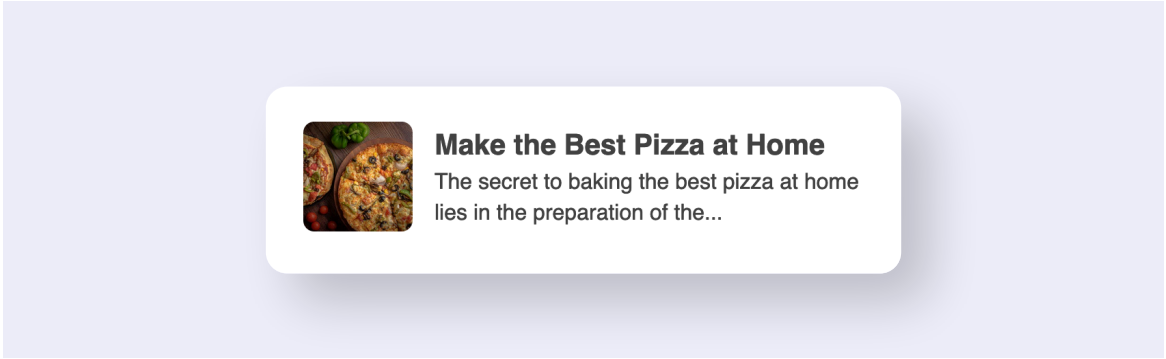
2. If there's less space AND `flex-shrink` is greater than 0, the item shrinks to occupy lesser than `100px`

By default, the value is `auto`, which means the size is auto-calculated based on the properties `width` or `height`.

The value can be in `%`, `px`, `em` or `rem` similar to the width and height properties. The value can also be a keyword such as `min-content`, `fill` and so on, which we will not be exploring in this book. Lets look at some more examples similar to the ones we have seen for `flex-grow` and `flex-shrink`, but this time we will use `flex-basis` instead of width or height.

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.



HTML

```
<div class="container">
  <div class="img-wrap">
    <img src=" ... ">
  </div>
  <div>
    ...
  </div>
</div>
```

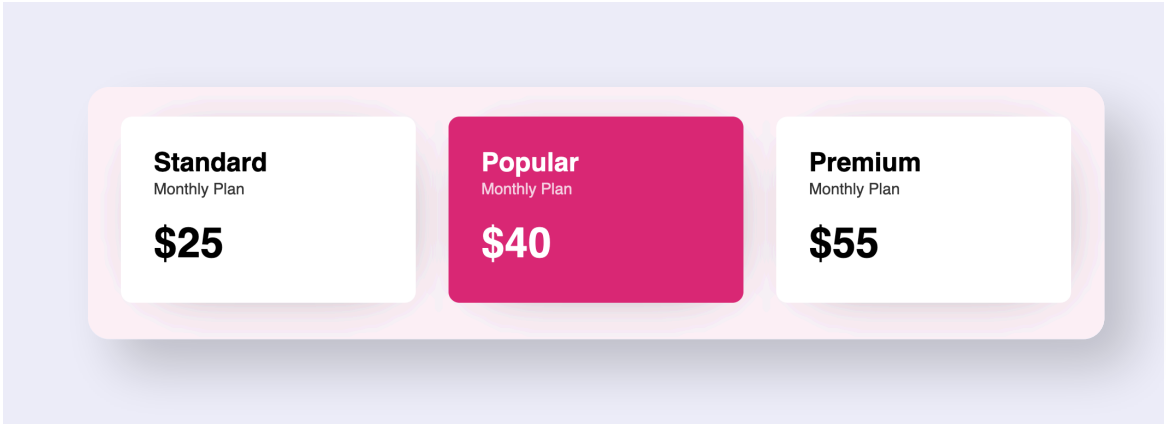
Solution

```
.container {
  display: flex;
}
.img-wrap {
  flex-basis: 5rem;
  flex-shrink: 0;
}
img {
  width: 100%;
}
```

▸ 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

HTML

```
<div class="container">
  <div class="plan">...</div>
  <div class="plan">...</div>
  <div class="plan">...</div>
</div>
```

Possible Solution

Set `flex-basis` as `33.33%` to all the flex items

```
.container {
  display: flex;
}
.plan {
  flex-basis: 33.33%;
}
```

▸ Working Demo

Note that though we set `flex-basis` to `33.33%`, the final width of each column is less than 33% of the parent because of the margin between the flex items. Each item shrinks by default to fit into the container.

Now there's a better solution to this:

Better Solution

```
.container {
  display: flex;
}
.plan {
  flex-basis: 0%;
  flex-grow: 1;
  flex-shrink: 1;
}
```

▸ Working Demo

This is better because if you add four blocks or two blocks instead of three, you don't have to change the `flex-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.

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`** property on the flex container. But this 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 properties `flex-grow`, `flex-shrink` and `flex-basis`, we can make use of a single `flex` shorthand property. In the previous example, you can replace all those three CSS rules with a single rule

```
flex: 1 1 0%;
```

Try replacing those 3 rules with this line in our previous example and notice that everything works the same.

Understanding `flex` Concept

The `flex` property is a shorthand property used on a **flex item** to combine the `flex-grow`, `flex-shrink` and `flex-basis`. It 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`
- 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`
- 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`

Three Values

The values must be in the following order:

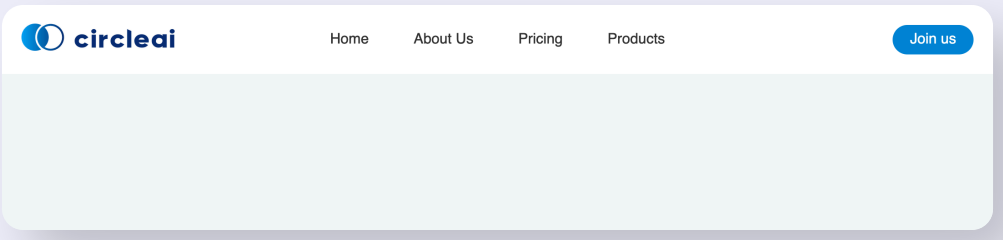
1. a `<number>` for `flex-grow`
2. a `<number>` for `flex-shrink`
3. a `<number with units>` for `<flex-basis>`

It is super hard to remember all of these at once, and the good news is - you don't have to! While 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.

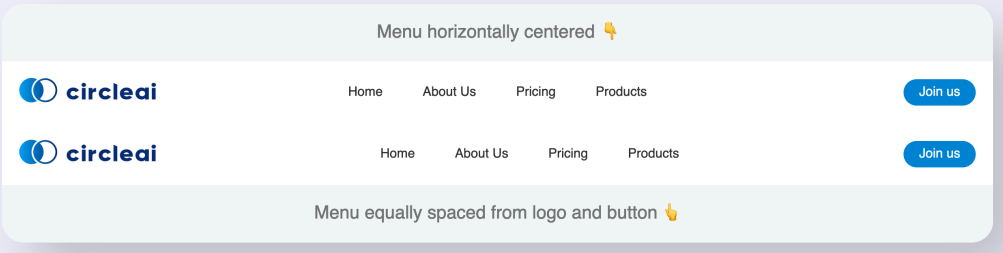
Now let's look at some examples using `flex` shorthand property.

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 you cannot simply use `justify-content: space-between` to achieve this. Look at the difference



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

▸ Try it out

HTML

```
<header>
  <a class="logo-wrap"> ... </a>
  <ul> ... </ul>
  <span class="btn-wrap"> ... </span>
</header>
```

Solution

If the elements `.logo-wrap` and `.btn-wrap` are of same width, then `justify-content: space-between` will help us achieve the desired result. Luckily, we can *make* them occupy the same widths with what we learnt in Example 8c

```
.logo-wrap, .btn-wrap {
  flex: 1 1 0%;
  /* Same as flex: 1 */
}
```

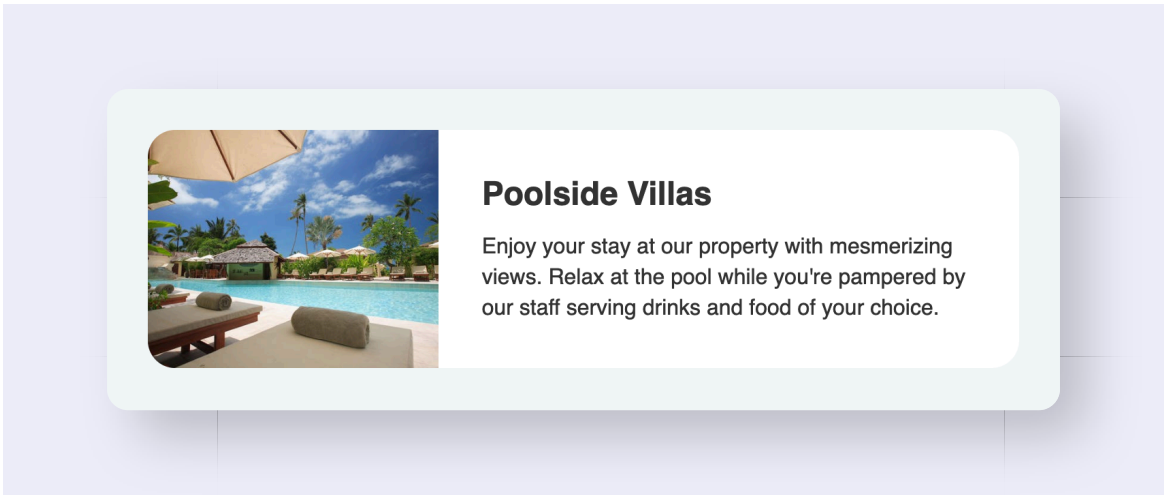
We are just one step away now. The button is aligned to the left of `.btn-wrap`. So add one more rule

```
.btn-wrap {
  text-align: right;
}
```

▸ 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` shorthand property.

▸ Try it out

HTML

```
<div class="container">
  <img ... >
  <div class="details"> ... </div>
</div>
```

Solution

```
img {
  flex: 1; /* Same as flex: 1 1 0% */
}
.details {
  flex: 2; /* Same as flex: 2 1 0% */
}
```

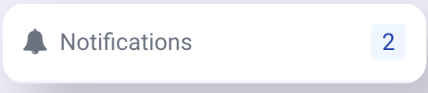
▸ Working Demo

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



HTML

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

If you can wrap the icon and text within another `div`, we can achieve this look by using `justify-content: space-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: 1` to the `.text` element. Here's another solution to simply push the `.count` element to the right.

Solution

```
.count {
  margin-left: auto;
}
```

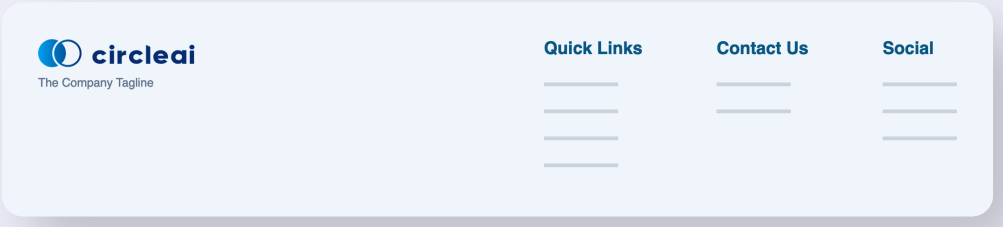
▸ Working Demo

The `margin` property 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 `margin: 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

HTML

```
<footer>
  <div class="footer-col"> ... </div>
  <div class="footer-col"> ... </div>
  <div class="footer-col"> ... </div>
  <div class="footer-col"> ... </div>
</footer>
```

Solution

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

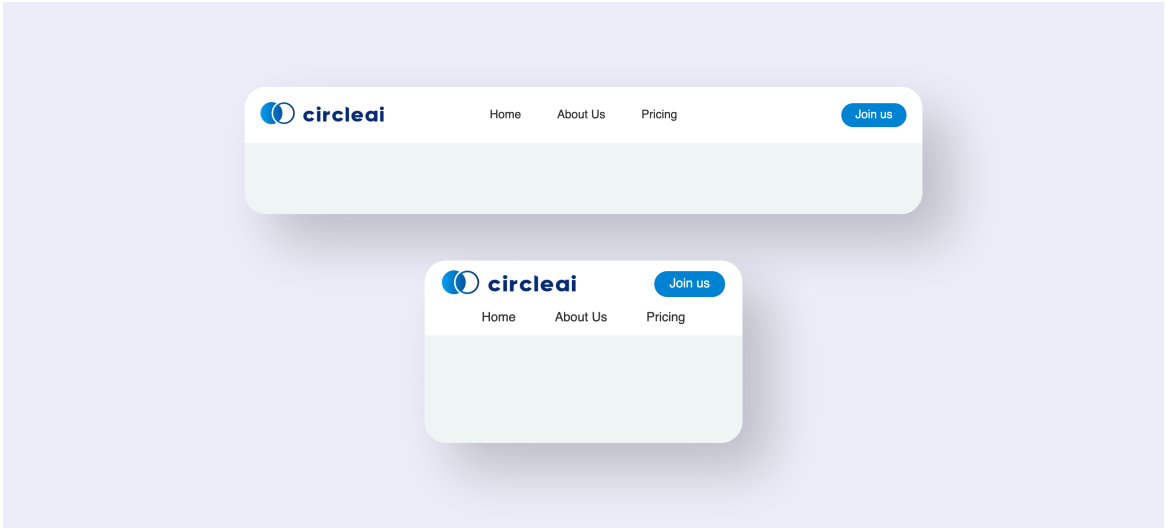
```
.footer-col:nth-child(2) {
  margin-left: auto;
}
```

▸ 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.



HTML

```
<header>
  <a class="logo-wrap"> ... </a>
  <ul> ... </ul>
  <span class="btn-wrap"> ... </span>
</header>
```

For smaller screens, we need to do three things:

1. Allow the flex items to `wrap`
2. Change the order of the flex items

3. Make the `ul` element occupy full width.

If you are aware of the `order` property, give this a shot.

▸ Try it out

Solution

```
@media screen and (max-width: 767px) {
  header {
    flex-wrap: wrap;
  }
  ul {
    order: 3;
    flex: 100%; /* flex-basis: 100% */
  }
}
```

▸ Working Demo

The only new property here is the **`order`** property.

Understanding `order` Concept

The `order` property is used on a **flex item** 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. Hence when we set `order: 3` to the `ul` element in the example above, only that element is removed from the normal flow and placed at the end.

Mobile First Approach

Here again, you can use the mobile first approach if you wish

HTML

```
<header>
  <a class="logo-wrap"> ... </a>
  <span class="btn-wrap"> ... </span>
  <ul> ... </ul>
</header>
```

Solution

```
header {
  /* Other flex styles here */
  flex-wrap: wrap;
}
ul {
  flex: 100%;
  text-align: center;
}
@media screen and (min-width: 768px) {
  header {
    flex-wrap: no-wrap;
  }
  span {
    order: 3;
  }
  ul {
    flex: auto;
    margin: 0;
  }
}
```

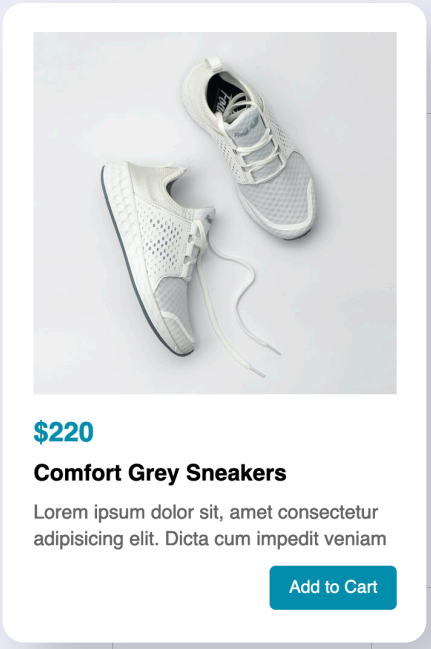
▸ Working Demo

Notice how we have used `flex: 100%` for the `ul` element and then changed it to `flex: auto` for larger screens. Also, here we used the `order` property on the `span` element to make it appear **last** on larger screens.

12 Align Self

Product Display Example 12a

This is a card component using `column` direction in flexbox. All the elements are stretched full width (along the cross axis) by default. But you want only the button to be pushed to the right instead of stretching full width.



▸ Try it out

HTML

```
<div class="container">
  <img ... >
  <span> ... </span>
  <h3> ... </h3>
  <p> ... </p>
  <button> ... </button>
</div>
```

Solution

```
.container {
  display: flex;
  flex-direction: column;
}
button {
  align-self: flex-end;
}
```

▸ Working Demo

We are using the property `align-self` for the `button` to align only that flex item along the cross axis. Remember, main axis and cross axis?

Understanding `align-self` Concept

The property `align-self` for a flex item is similar to the `align-items` property. This property overrides the `align-items` property of its parent container. Note that:

1. `align-self` is applied to a flex item, whereas `align-items` is applied to a flex container

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

The values are also similar to `align-items.` Following values can be given to `align-self`:

`stretch`

The item is stretched to fill the container along the cross axis

`center`

The item is placed at the center of the container

`flex-start`

The item is placed at the beginning of the container *(at the top for `row` direction and at the left for `column` direction)*

`flex-end`

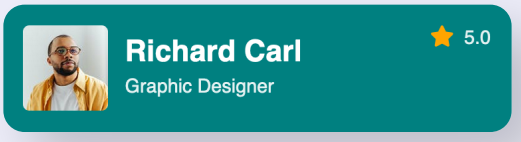
The item is placed at the end of the container *(at the bottom for `row` direction and at the right for `column` direction)*

`baseline`

The item is positioned such that the base aligns to the end of the container *(applies only for `row` direction)*

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

HTML

```
<div class="container">
  <img ... >
  <div> ... </div>
  <div class="rating"> ... </div>
</div>
```

Solution

You need two CSS rules - one for pushing it to the right *(alignment along main axis)* and another for aligning it at the top *(alignment along cross axis)*.

```
.container {
  display: flex;
  align-items: center;
}
.rating {
  margin-left: auto; /* Alignment along main axis */
  align-self: flex-start; /* Alignment along cross axis | Overrides line 3 for this item */
}
```

▸ Working Demo

This example might help you understand why we have a property `align-self` 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 Flex Flow

We have another shorthand property `flex-flow` used on a flex container that combines `flex-direction` and `flex-wrap`.

```
flex-flow: <flex-direction> <flex-wrap>
```

Example: `flex-flow: row wrap`

Let's try this on Example 3b

HTML

```
<div class="logos">
  <img ... >
  <img ... >
  <img ... >
  <img ... >
</div>
```

Solution

```
.logos {
  display: flex;
  justify-content: space-around;
  flex-flow: row wrap;
}
```

▸ Working Demo

14 Align Content

Full Page Testimonials Section Example 14a

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 `align-items: center` does. But no. That works only for single row flex items.

▸ Try it out

HTML

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

```
.container {
  display: flex;
  flex-flow: row wrap;
  justify-content: center;
  align-content: center;
}
```

▸ Working Demo

All we need to do is use the `align-content` property instead of `align-items`.

Understanding `align-content` Concept

The property `align-content` is 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. Following values can be given to `align-content`:

`flex-start`

The items are packed at the beginning of the container

`flex-end`

The items are packed at the end of the container

`center`

The items are packed at the center of the container

`space-between`

The rows / columns are spaced out as much as possible with first line at the beginning and last line at the end

`space-around`

Space at the beginning and the end are half as much as space between the lines

`space-evenly`

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 😀

15 Inline Flex

Social Media Icons Example 15a

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 `display-flex` to the circle and center using `justify-content` and `align-items`. But that leaves us with the circles stacked one below the other, instead of next to each other

▸ Try it out

HTML

```
<div class="wrapper">
  <a class="icon" href="#">
    <i class="fa fa-twitter"></i>
  </a>
  <a class="icon" href="#">
    <i class="fa fa-linkedin"></i>
  </a>
  <!-- More Icons -->
</div>
```

Solution

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

```
.icon {
  display: inline-flex;
  justify-content: center;
  align-items: center;
}
```

▸ Working Demo

Understanding `inline-flex` Concept

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

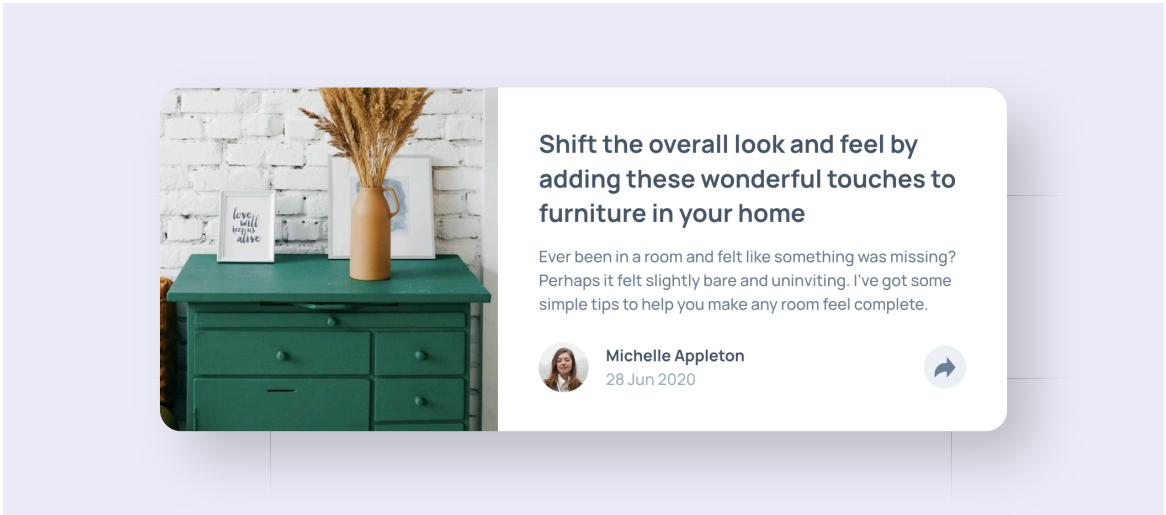
Let's look at some comprehensive examples each involving multiple flexbox containers in each component.

Comprehensive Examples for Flexbox

Article Preview Example 16a

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 image and text block being flex items. 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.

This component needs to to made responsive too, by changing the outer flex container's direction to `column` for smaller screens. On the other hand, you can try the mobile first approach too.

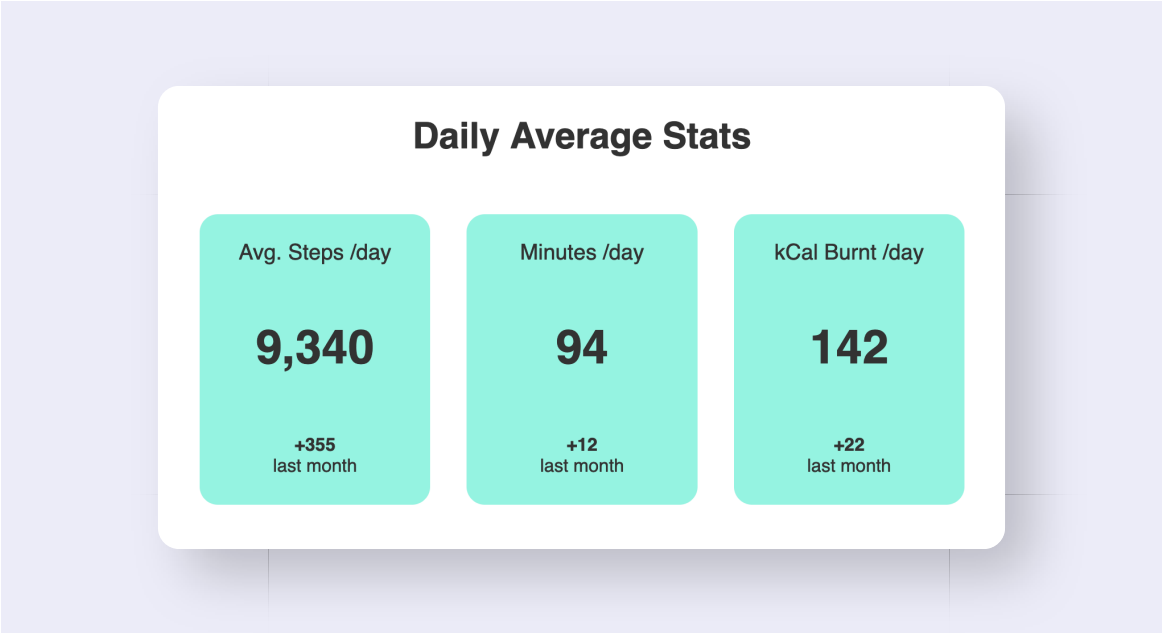
▸ Try it out

HINT: Use the properties `flex`, `align-items`, `flex-direction` and auto margins.

▸ Working Demo

Fitness Report Example 16b

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 properties `flex-direction`, `flex`, `justify-content`, `min-width` and `max-width`

▸ Working Demo

Tweet Example 16c

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 properties `align-items`, `justify-content` and auto margins.

▸ Working Demo

Whoa! This really completes 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.

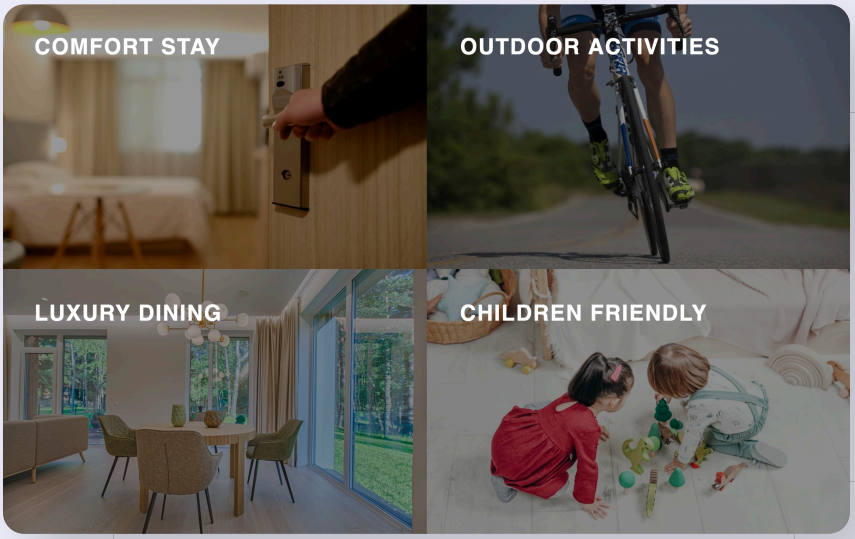

Grid

17 Display Grid & Grid Template Columns

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

Full Page Gallery Example 17a

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

HTML

```
<div class="container">
  <div class="item">...</div>
  <!-- Three more items -->
</div>
```

Solution

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

```
.container {
  display: grid;
  grid-template-columns: 50% 50%;
}
```

▸ Working Demo

Do you see that? 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 properties 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 `display: grid` makes the `.container` element a grid container. But this CSS rule alone does not make any difference because it creates one column by default. We need the next property `grid-template-columns` to specify the number of columns and width of each.

Understanding `grid-template-columns` Concept

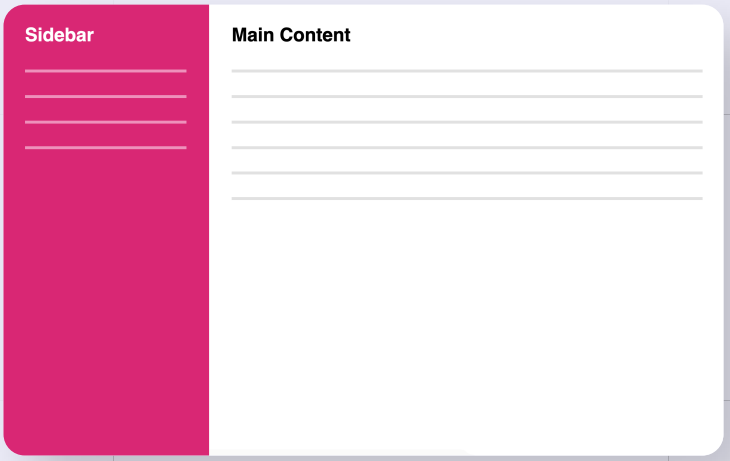
The property `grid-template-columns` is used to specify how many columns you need and of what size each. This is done by specifying the widths of each column in `%`, `px`, `rem` or any valid value for width separated by spaces. The number of individual values you specify will be the number of columns created. The simplest example was the previous one.

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

This created two columns of 50% width each. But the syntax gets more complex for complex layouts. Let's look into each of them with appropriate examples.

Layout with Sidebar Example 17b

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.



HTML

```
<div class="container">
  <div class="sidebar">...</div>
  <div class="main">...</div>
</div>
```

Solution

We have 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`.

```
.container {
  display: grid;
  grid-template-columns: 22rem 1fr;
}
```

▸ Working Demo

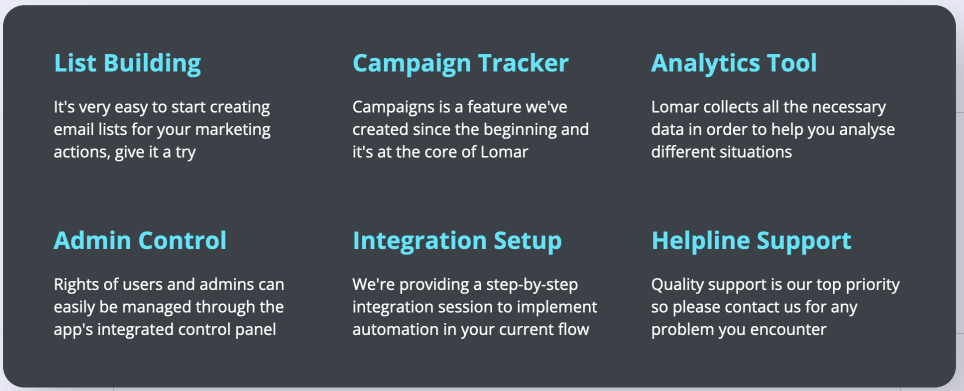
The `fr` Unit

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 17c**

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? Bonus points if you use the `fr` unit.

Try it out

HTML

```
<div class="container">
  <div class="item">...</div>
  <!-- Five more items here -->
</div>
```

Solution

We need three columns of equal width. The `fr` unit helps us distribute the remaining space (which is all the space in this example) proportionately.

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

So, you can repeat the `fr` 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

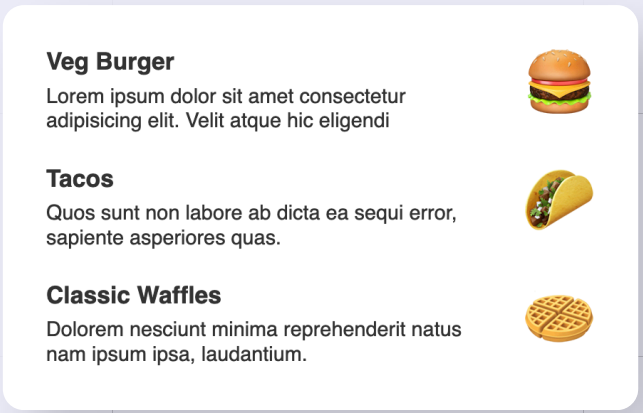
```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```

▸ Working Demo

With this new knowledge, can you modify your solution to Example 17a by including `fr` units?

Quick Bites Menu Example 17d

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 would 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

HTML

```
<div class="container">
  <div class="item">...</div>
  <span class="icon">...</span>
  <div class="item">...</div>
  <span class="icon">...</span>
  <div class="item">...</div>
  <span class="icon">...</span>
</div>
```

Solution

We need two columns, so we can use `grid-template-columns` to specify two values separated by spaces. 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.

```
.container {
  display: grid;
  grid-template-columns: 1fr auto;
}
```

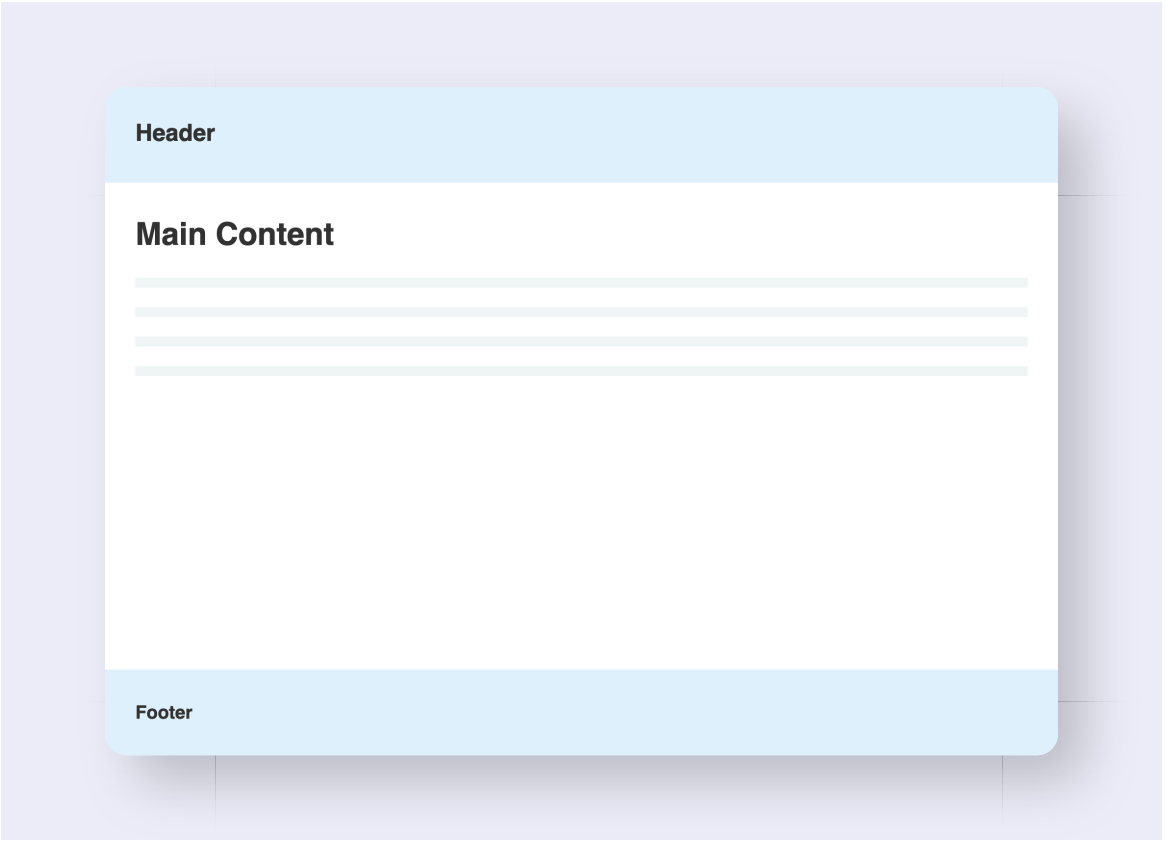
▸ Working Demo

So far we used fixed units, percentage values and the `auto` keyword for specifying `grid-template-columns`. There are few more options that we'll cover under the Advanced Grid Template Values topic.

18 Grid Template Rows

Sticky Footer with Grid Example 18a

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-direction: column` to create this.



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

▸ Try it out

HTML

```
<div class="container">
  <header> ... </header>
  <div class="main"> ... </div>
  <footer> ... </footer>
</div>
```

Solution

Look at the example and observe that we have a single column but multiple rows. This can be specified using `grid-template-rows` instead of `grid-template-columns`.

```
.container {
  min-height: 100vh;
  display: grid;
  grid-template-rows: auto 1fr auto;
}
```

▸ 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 content you want, you need to add it within the `.main` div.

Understanding `grid-template-rows` Concept

The property `grid-template-rows` 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***.

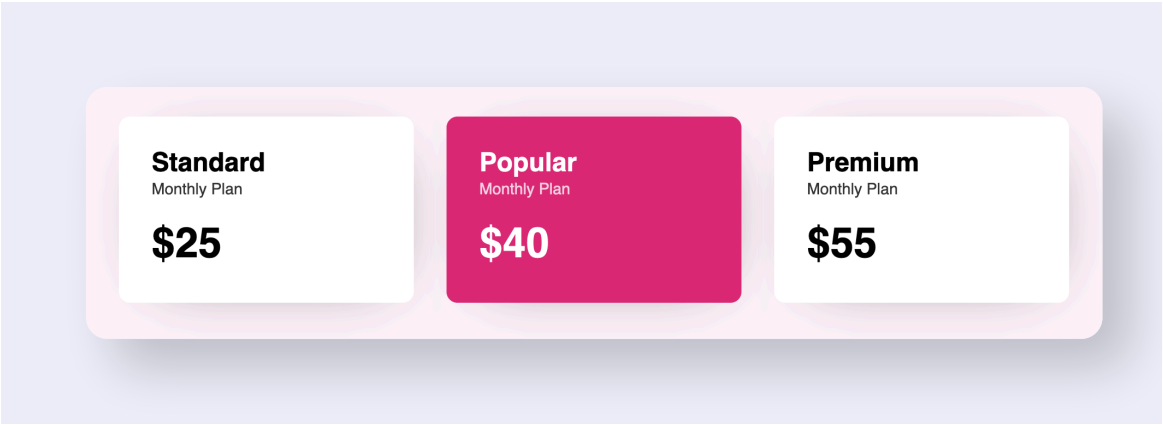
Similarly in our above example, one column is automatically created that occupies full width by default. We'll look into implicit grids and their sizing in a while. We will also look at more examples of `grid-template-rows` after covering a few more concepts. For now, I'm sure you have a basic idea.

And of course, you can use both `grid-template-columns` and `grid-template-rows` at the same time.

19 Gap

Pricing Plans with Grid Example 19a

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

HTML

```
<div class="container">
  <div class="plan">...</div>
  <div class="plan">...</div>
  <div class="plan">...</div>
</div>
```

Solution

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

▸ Working Demo

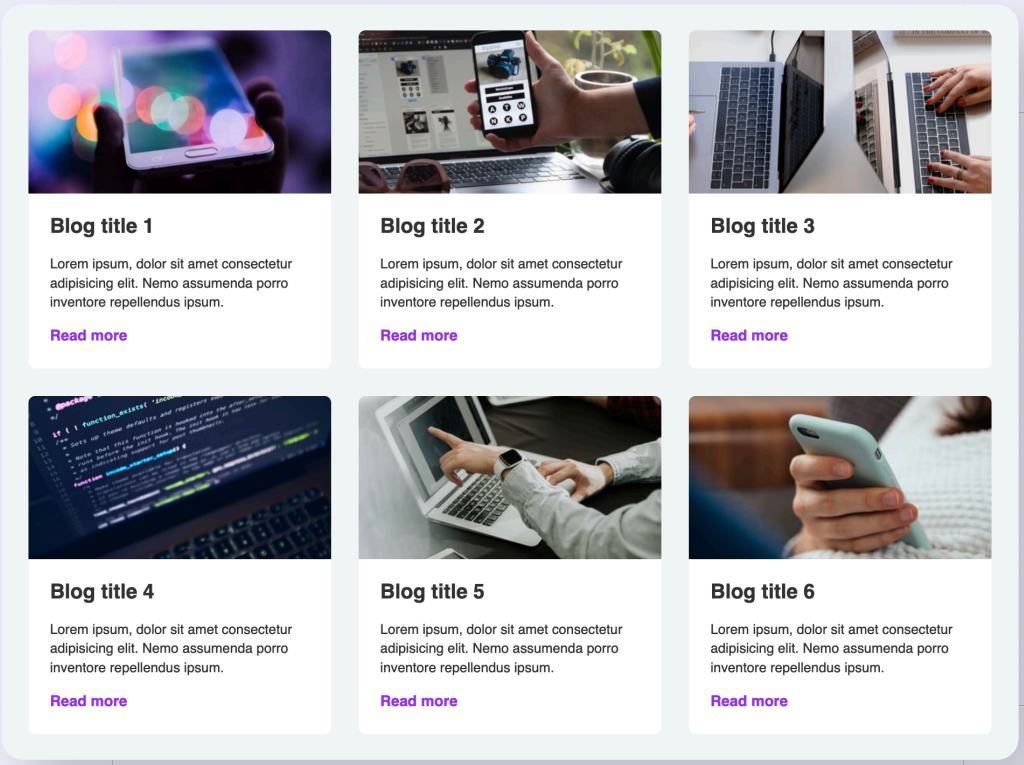
Understanding `column-gap` Concept

The property `column-gap` sets the size of the gap (also known as gutters) in 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.

The value of `column-gap` must be a non-negative value specified in `px`, `%`, `rem` or any valid length unit.

Blog Posts Display Example 19b

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 `column-gap` and a similar property `row-gap`.

▸ Try it out

HTML

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

Responsive Solution

Using the mobile first approach, we create only one column of grid first. At `540px` breakpoint, we change that to two columns and at `767px`, we change it to three columns. We also add a `column-gap` and `row-gap` of `2rem` each.

```
.container {
  display: grid;
  grid-template-columns: 1fr;
  column-gap: 2rem;
  row-gap: 2rem;
}

@media screen and (min-width: 540px) {
  .container {
    grid-template-columns: repeat(2, 1fr);
  }
}

@media screen and (min-width: 767px) {
  .container {
    grid-template-columns: repeat(3, 1fr);
  }
}

```

▸ Working Demo

These gap properties make it very easy to create spacing only between the items and not around them. If we had to use `margin` property 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 these properties into one single `gap` property.

Better Solution

```
.container {
  /* Other styles here */
  gap: 2rem;
}
```

Understanding `row-gap` Concept

The property `row-gap` sets the size of the gap or gutters in between the 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 value of `row-gap` must be a non-negative value specified in `px`, `%`, `rem` or any valid length unit.

Understanding `gap` Concept

The property `gap` is a shorthand property for `row-gap` and `column-gap`. It can specified with one or two values.

One Value

If we specify only one value for this property, like in our previous example, that value is assigned to both `row-gap` and `column-gap`.

Two Values

The first value is `row-gap` and the second one is `column-gap`.

We will start using these gap properties in most of our upcoming examples.

20 Justify Content

Featured Logos in a Grid Example 20a

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.

▸ 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.

HTML

```
<div class="container">
  <img ... >
  <!-- Seven more img elements -->
</div>
```

Solution

We need to first use `grid-template-columns` to create four columns. Set the column widths as `auto` to size them based on the content *(the image width)*. That's when we can use the property `justify-content` to space them out. This is similar to what we did with flexbox.

```
.container {
  display: grid;
  grid-template-columns: repeat(4, auto);
  gap: 3rem;
  justify-content: space-between;
}
```

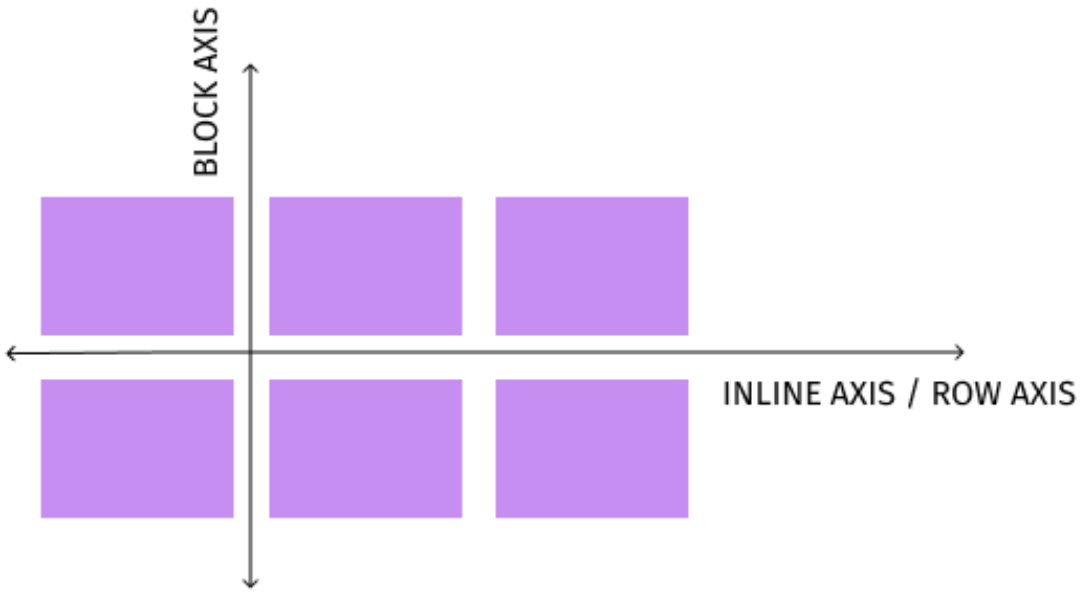
▸ Working Demo

This is not yet responsive. We will do that once we cover few more concepts, and you will see why. And to notice the difference between flex and grid, change the first two CSS rules above to `display: flex` and `flex-wrap: wrap`.

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 property **`justify-content`** is used to control spacing of the grid items along the *inline/row axis* - which is the horizontal direction. `space-between` is one of the values we just used.

Following values can be given to `justify-content`:

`start`

All columns are placed at the beginning of the container

`end`

All columns are placed at the end of the container

`center`

All columns are placed at the center

`space-between`

All columns are spaced out as much as possible with first item at the beginning and last item at the end

`space-around`

Space on the left and right are half as much as space between the items

`space-evenly`

Space on the left, right and between the items are same

`stretch`

If the 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 set a value to the `justify-content` property.

In your browser (Chrome or Firefox), use the inspector tool on the CodePen demo 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-template-columns` 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).

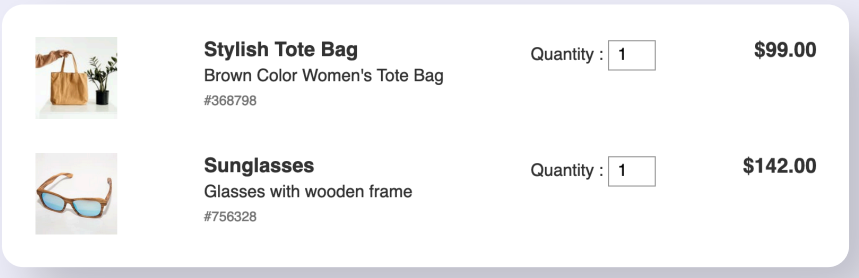
But when we set `justify-content: space-between`, this happens:

The grid cells now take up only as much width required. The remaining space is added to the grid lines 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 property `justify-content` has no effect!

Shopping Cart Summary Example 20b

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-content` here too, to space out the columns.



▸ Try it out

HTML

```
<div class="container">
  <img ... >
  <div class="desc"> ... </div>
  <div class="qty"> ... </div>
  <div class="price"> ... </div
  <img ... >
    ...
</div>
```

Solution

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

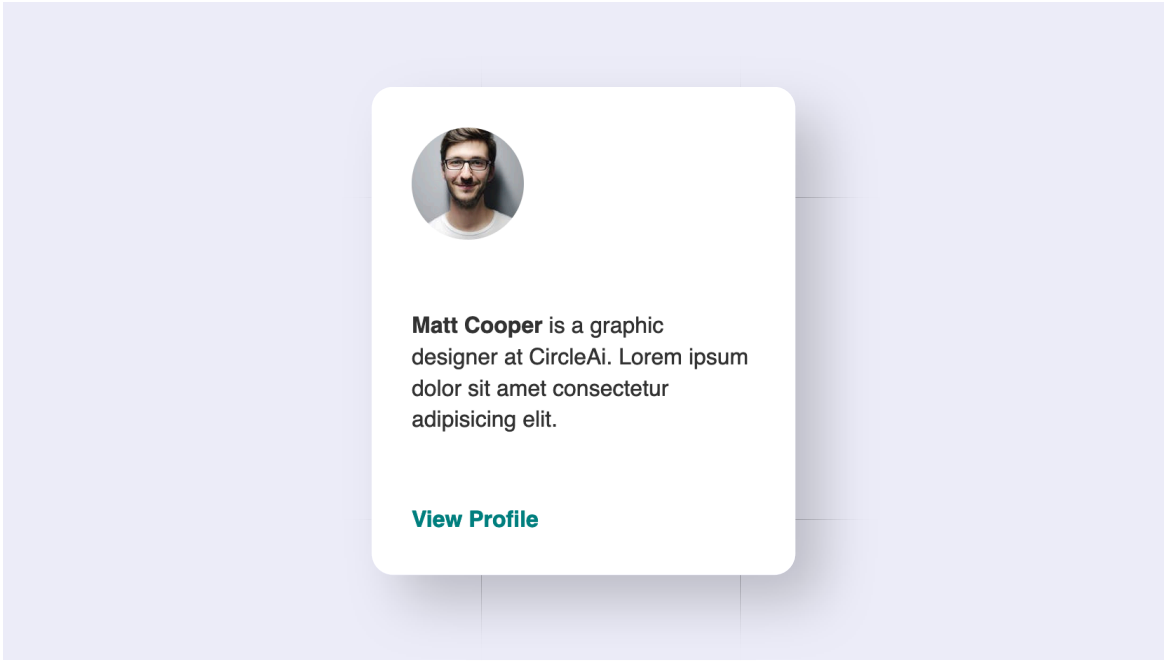
```
.container {
  display: grid;
  grid-template-columns: repeat(4, auto);
  gap: 2rem 1rem;
  justify-content: space-between;
}
.price {
  text-align: right;
}
```

▸ Working Demo

21 Align Content

Profile Card with Bio & Link Example 21a

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 line shorter.



▸ Try it out

HTML

```
<div class="card">
  <img ... >
  <p> ... </p>
  <a> ... </a>
</div>
```

Solution

For this one, we need only one column, so adding `display: 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 `align-content`.

```
.card {
  display: grid;
  align-content: space-between;
}
```

▸ Working Demo

Understanding `align-content` in Grid Concept

We have seen `align-content` with respect to flexbox to control spaces between multiple lines of wrapped items along the *cross axis*. In Grid, the property **`align-content`** is used to control spacing of the grid items along the *block axis* - which is the vertical direction. `space-between` is one of the values we just used.

All the values that can be assigned to `justify-content` can be used for `align-content` too. They give the same output in the vertical direction.

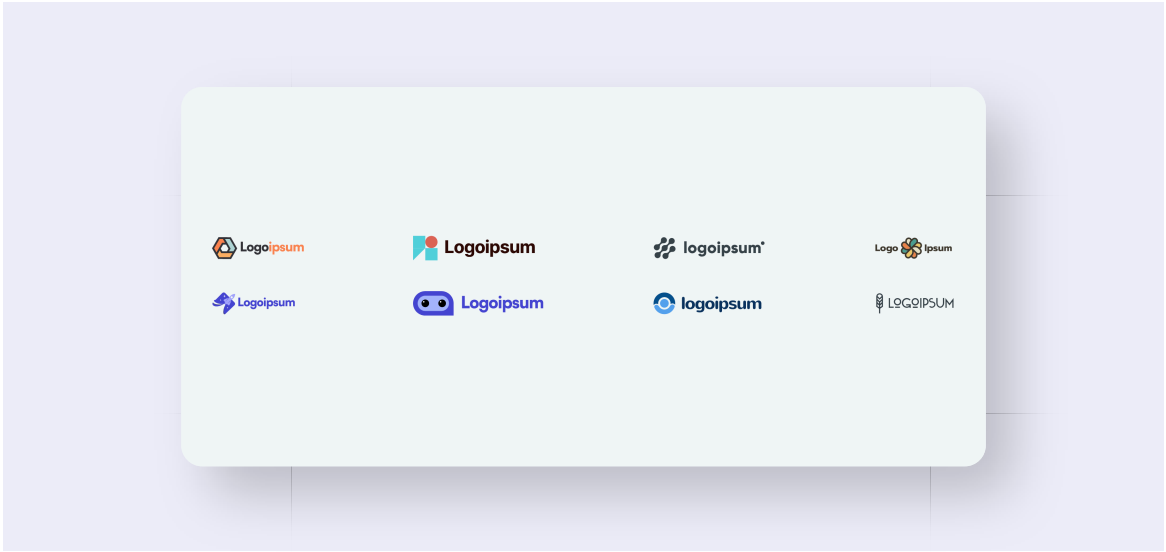
Note : This property has an effect only when

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

22 Place Content

Featured Logos Center of Page Example 22a

In Example 20a, we looked at displaying logos in a grid and spacing them horizontally with `justify-content`. 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.

▸ Try it out

HTML

```
<div class="container">
  <img ... >
  <!-- Seven more img elements -->
</div>
```

Solution

There are surely other ways of doing it, but the simplest solution is to use `align-content` along with `justify-content`. And there's a shorthand property combining these two:

```
.container {
  min-height: 100vh;
  /* Other styles here */
  place-content: center space-between;
}
```

▸ Working Demo

We will be making some more changes to this example soon.

Understanding `place-content` in Grid Concept

The property `place-content` is a shorthand property in Grid that allows you to control the spacing of grid items along both the block and inline axes at once. It can specified with one or two values.

One Value

If we specify only one value for this property, like in our previous example, that value is assigned to both `justify-content` and `align-content`.

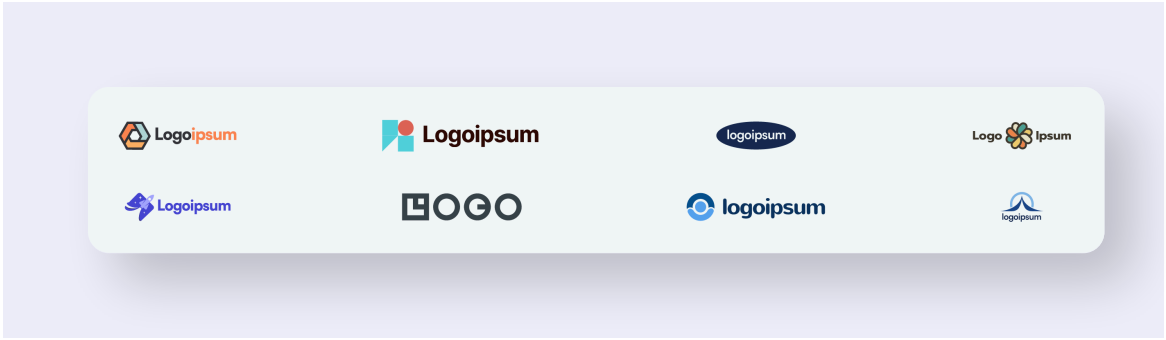
Two Values

The first value is `align-content` and the second one is `justify-content`.

23 Justify Items

Featured Logos of Different Widths Example 23a

We're back with the same example with 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

HTML

```
<div class="container">
  <img ... >
  <!-- Seven more img elements -->
</div>
```

Solution

There's one new property to our rescue.

```
.container {
  display: grid;
  grid-template-columns: repeat(4, auto);
  justify-content: space-between; /* To space out the columns horizontally */
  justify-items: center; /* To center align the logos horizontally within each column */
}
```

▸ 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 property `justify-items` allows us to **horizontally** align the content *within* the columns, while the previous property `justify-content` allows us to horizontally space out the entire columns.

Following values can be used for `justify-items`:

`start`

All items are placed at the beginning of their columns

`end`

All items are placed at the end of their columns

`center`

All items are placed at the center of the columns

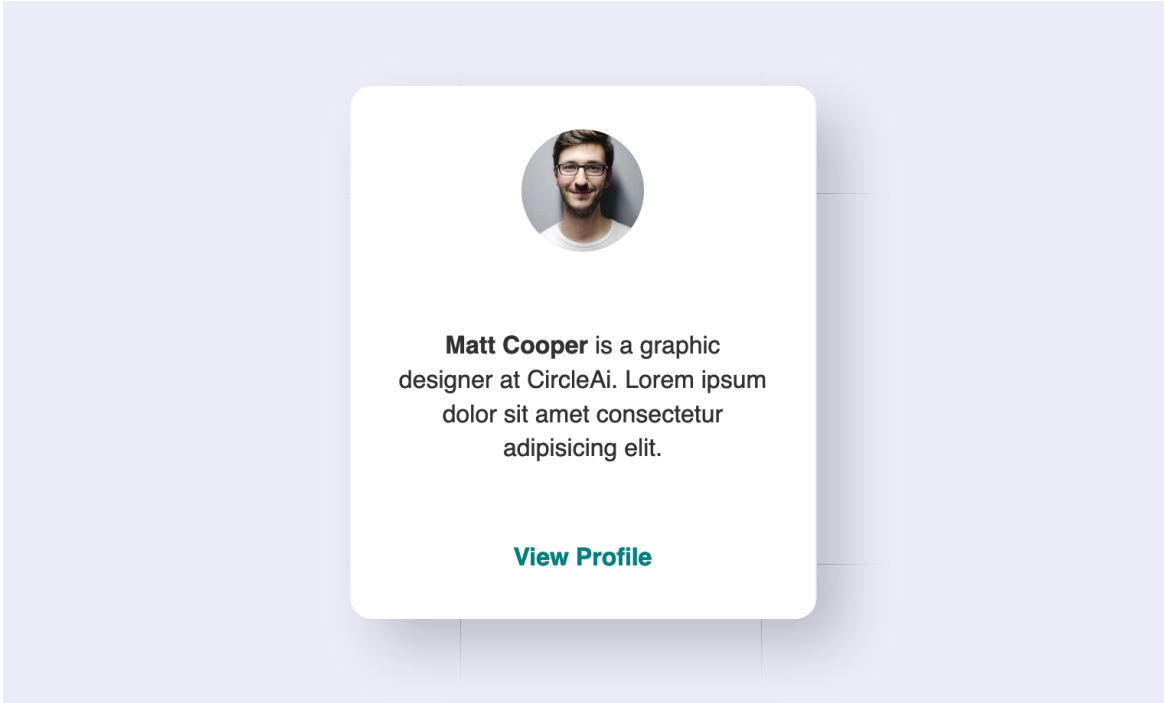
`stretch`

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 23b

We are revisiting Example 21a, this time making everything center aligned horizontally.



▸ Try it out

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

HTML

```
<div class="card">
  <img ...>
  <p>...</p>
  <a>...</a>
</div>
```

Solution

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

```
.card {
  display: grid;
  align-content: space-between;
  justify-items: center;
  text-align: center; /* For the bio */
}
```

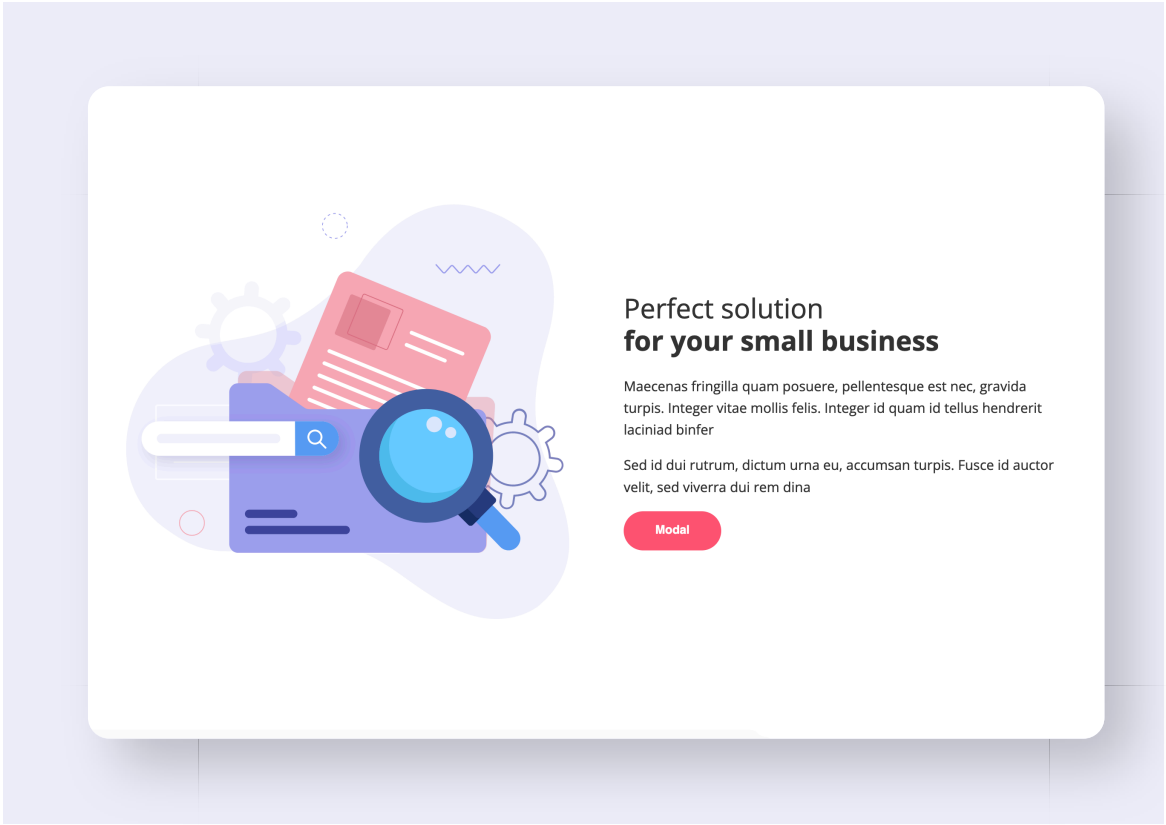
▸ Working Demo

24 Align Items

Image and Text Section Example 24a

Example from Inovatik Template

One very common section in web pages is an image on the left half and text on right half of a webpage. 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

HTML

```
<section class="container">
  <img ... >
  <div> ... </div>
</section>
```

Solution

Using `grid-template-columns` we can create two equal sized columns. Using `gap`, we can add some spacing between them. To center align the image and text, we need to use the `align-items` property, very similar to flexbox.

```
.container {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  gap: 4rem;
  align-items: center;
}
```

▸ 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

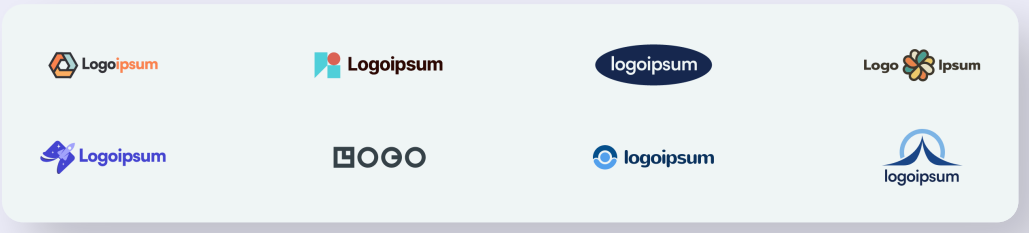
top.

The property `align-items` allows us to **vertically** align the content *within* the rows, while the previous property `align-content` allows us to vertically space out the entire rows.

All the values that can be assigned to `justify-items` can be used for `align-items` too. They give the same output in the vertical direction.

Featured Logos of Different Heights Example 24b

Let's look at the same example once again. So far, we used a fixed `height` of `2.4rem` for all the logos. Now we'll remove the `height` and instead add a `max-width` of `10rem`. 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 23a, this time we also need to vertically center align them in each row.



▸ Try it out

HTML

```
<div class="container">
  <img ... >
  <!-- Seven more img elements -->
</div>
```

Solution

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

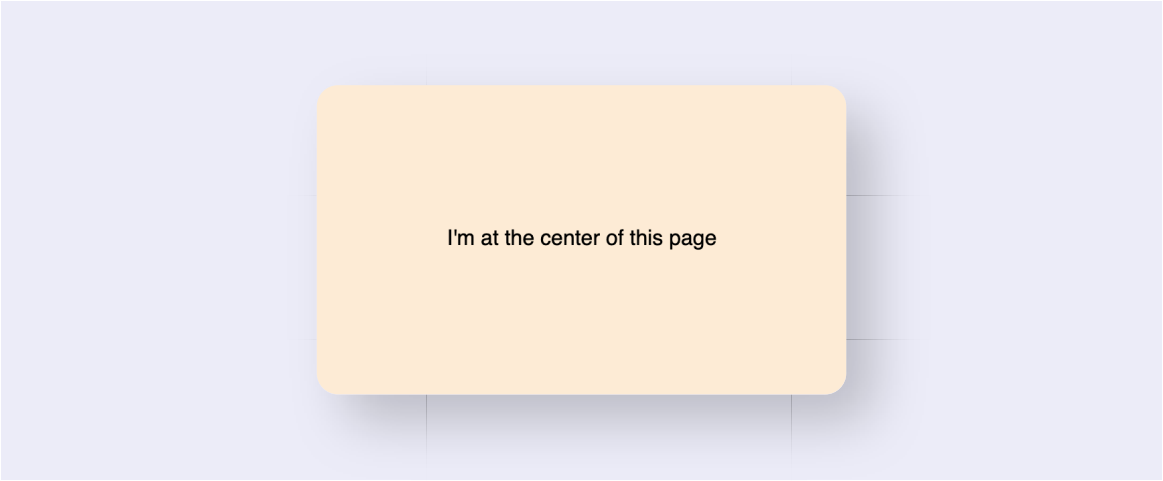
```
.container {
  display: grid;
  grid-template-columns: repeat(4, auto);
  justify-content: space-between; /* To space out the columns horizontally */
  justify-items: center; /* To center align the logos horizontally within each column */
  align-items: center; /* To center align the logos vertically within each row */
}
```

▸ **Working Demo**

25 Place Items

Center a div using Grid Example 25a

We have already seen how easy it is to center a div using flexbox. With grid, it's one more line shorter.



HTML

```
<div class="container">
  <div class="item">
    ...
  </div>
</div>
```

Solution

You can either use the previous two properties `justify-items` and `align-items` along with grid. Or you can combine both these properties using the shorthand property `place-items`.

```css
.container {
  display: grid;
  place-items: center;
}
```

▸ Working Demo

Understanding `place-items` Concept

The property `place-items` is a shorthand property in Grid that allows you to align items along both the block and inline directions at once. It can specified with one or two values.

One Value

If we specify only one value for this property, like in our previous example, that value is assigned to both `justify-items` and `align-items`.

Two Values

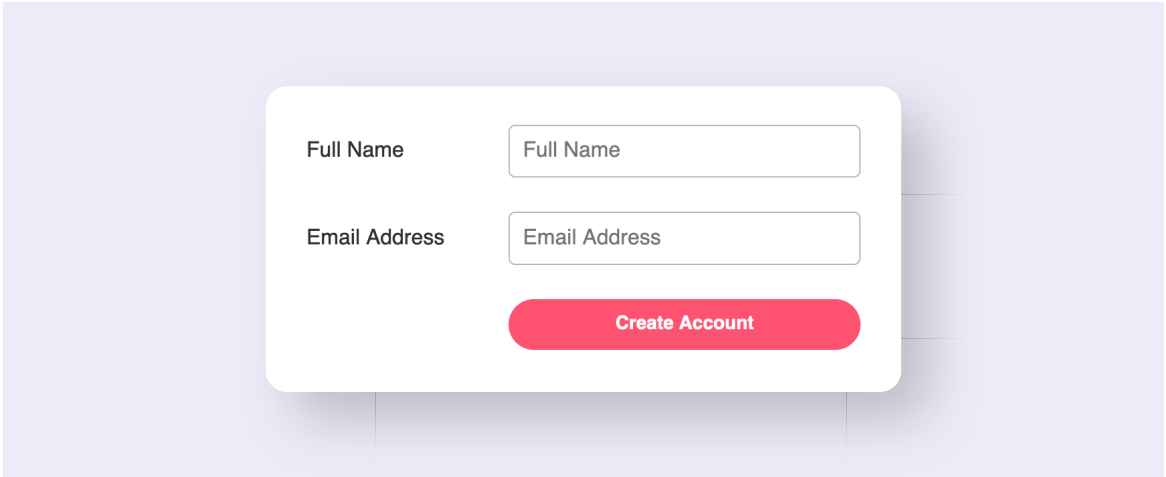
The first value is `align-items` and the second one is `justify-items`.

Now you can go back to Example 24b to replace two properties with `place-items`.

26 Grid Column

Horizontal Form Example 26a

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-template-columns` to create two columns. 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

HTML

```
<form>
  <label></label>
  <input .. />
  <label></label>
  <input .. />
  <button ...>Create Account</button>
</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 CSS solution available.

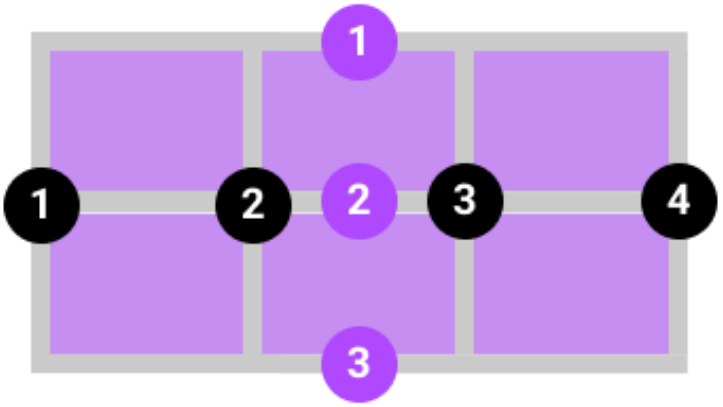
```
form {
  display: grid;
  grid-template-columns: auto 1fr;
  /* gap and align properties here */
}
button {
  grid-column-start: 2;
}
```

▸ Working Demo

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

Understanding `grid-column-start` Concept

Before we learn more about the `grid-column-start` property, we need to learn about **grid lines**. When you define a grid using `grid-template-columns` and/or `grid-template-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 properties we saw so far are applied to the grid container. Now, we will see a few that are applied to the grid items. The property `grid-column-start` is one of them. It specifies the item's start position.

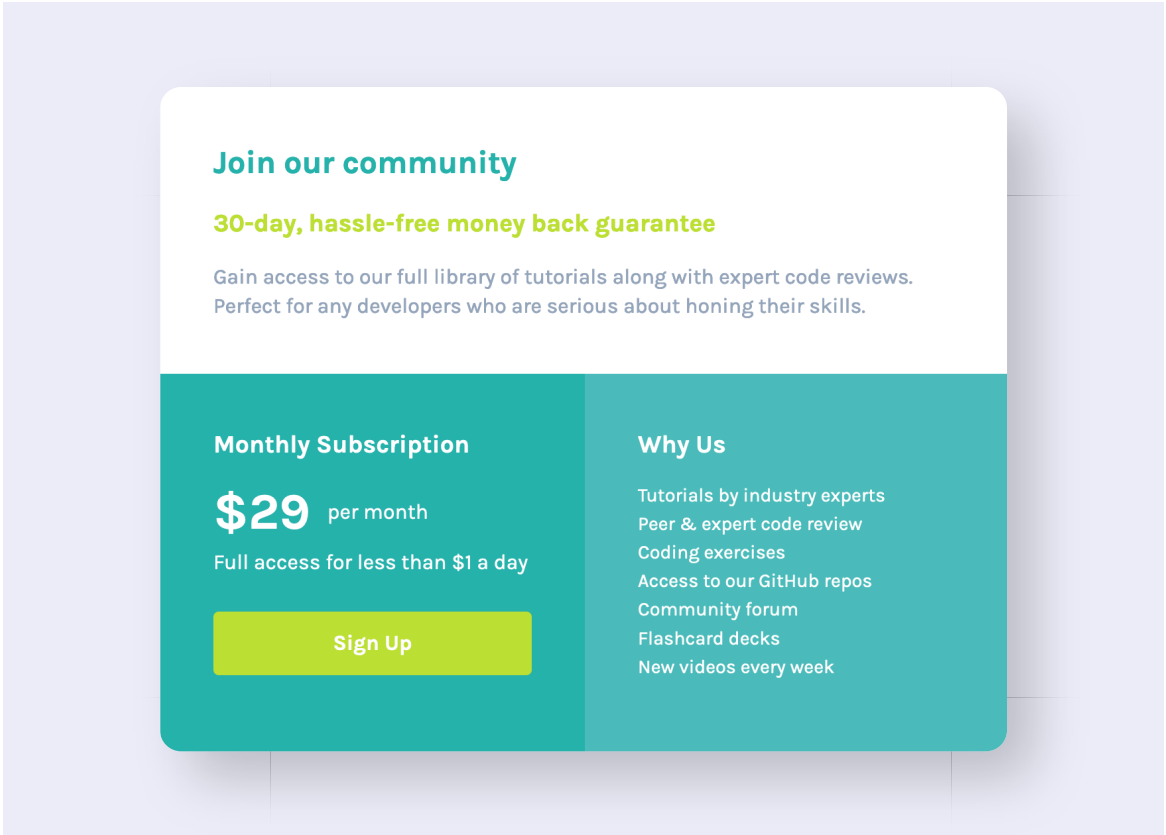
The value is usually a positive integer specifying a column line. You can also use a negative integer, in which case it starts counting the lines in reverse, starting from `-1`.

In the above example, we set this property to `2` because we wanted the button to start from column line 2. Now observe that if you set it to 3 or -1, it creates a new column and everything gets messed up. Hence you need to be careful with this value.

Single Price Grid Component Example 26b

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.



You can do this with the property `grid-column-start` that we just saw along with another property `grid-column-end`. We will use the mobile-first approach and make changes only to the desktop version.

▸ Try it out

HTML

```
<div class="container">
  <div class="component-header">...</div>
  <div class="subscription">...</div>
  <div class="why">...</div>
</div>
```

Solution

We have one column to start with and above the breakpoint `620px` we change it to two columns and make the element `.component-header` start from column line 1 and end at column line 2.

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

@media screen and (min-width: 620px) {
  .container {
    grid-template-columns: 1fr 1fr;
  }
  .component-header {
    grid-column-start: 1;
    grid-column-end: 3;
  }
```

▸ Working Demo

Let's learn about `grid-column-end` and few more ways of getting the same result.

Understanding `grid-column-end` Concept

The property `grid-column-end` is another grid item's property. It specifies the item's end position. The value is usually a positive integer specifying a column line. You can also use a negative integer, in which case it starts counting the lines in reverse, starting from `-1`.

In the above example, we set this property to `3` because we wanted that item to end at column line 3. You can also use the negative value `-1` which means the last column line.

Both the properties `grid-column-start` and `grid-column-end` can also be specified along with a `span` keyword.

Example: `grid-column-start: span 2` or `grid-column-end: span 2`

So, another solution to the previous example can be:

```
grid-column-start: 1;
grid-column-end: span 2;
```

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.

OR

```
grid-column-start: span 2;
grid-column-end: 3
```

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.

Now, we can also combine these two properties into a shorthand `grid-column` property.

With this new knowledge, let's re-visit the Horizontal Form Example and make it responsive keeping the desktop version as it is and adding media queries for screen sizes lesser than `530px`

Responsive Solution

```
1  form {
2    display: grid;
3    grid-template-columns: auto 1fr;
4    /* gap and align properties here */
5  }
6  button {
7    grid-column-end: -1;
8  }
9  @media screen and (max-width: 530px) {
10   form {
11     grid-template-columns: 1fr;
12   }
13 }
```

▸ Working Demo

In the earlier example, we used `grid-column-start: 2`. Here, we have replaced it with `grid-column-end: -1`. That's because, if we retain `grid-column-start: 2`, even if we change the number of columns to 1 on line 11, we still get two columns! Try using `grid-column-start: 2` in the above link and see for yourself. But why is that?

We have created one column explicitly using `grid-template-columns: 1fr`. But the CSS rule `grid-column-start: 2` creates an implicit second column because it assumes you want a second column. However if we use `grid-column-end: -1`, since the value `-1` just represents the last column line, there's no problem.

Next, let's look at the shorthand property `grid-column` that combines both the start and end properties.

Understanding `grid-column` Concept

The `grid-column` property takes one value or two values separated by forward slash `/`.

One Value

If we specify only one value for this property, that value is `grid-column-start`.

Two Values

The first value is `grid-column-start` and the second one is `grid-column-end`.

So we have three more solutions to Example 25b - Single price grid component:

```
grid-column: 1 / 3;
```

OR

```
grid-column: 1 / span 2;
```

OR

```
grid-column: span 2 / 3;
```

Try them out before you proceed to the next example.

Page Layout with Grid Example 26c

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

HTML

```
<body>
  <header> ... </header>
  <div class="sidebar"> ... </div>
  <div class="main"> ... </div>
  <footer> ... </footer>
</body>
```

Solution

There are three things we need to do:

1. We have a fixed width sidebar, so the `grid-template-columns` will have one fixed width value and `1fr`.
2. We will have to use `grid-template-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`. 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 `grid-column` to start with column line 1 and end at the last column line.

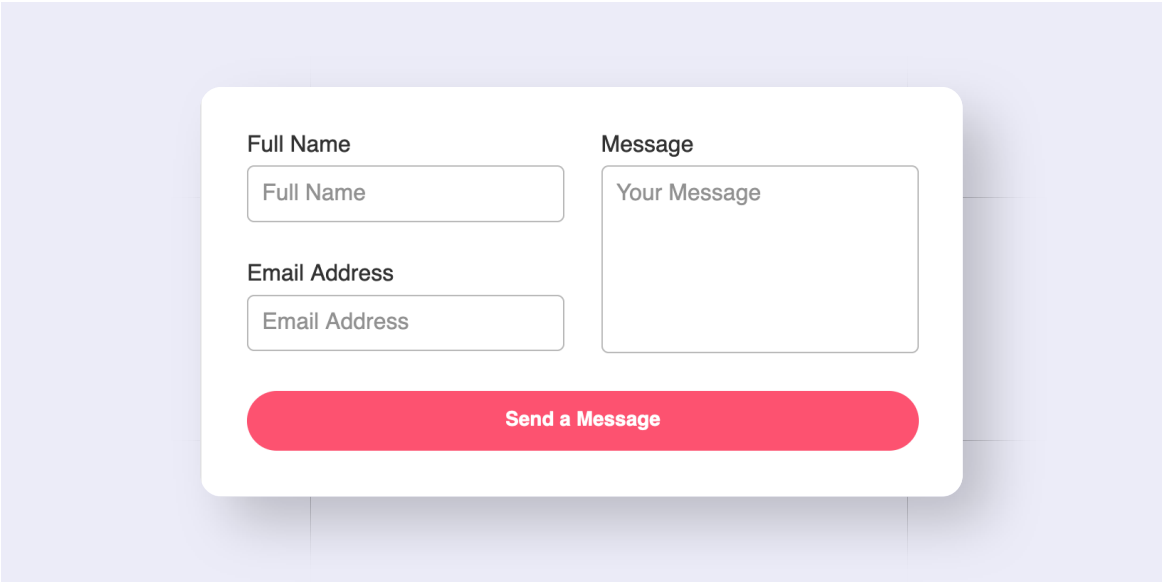
```
body {
  min-height: 100vh;
  grid-template-columns: 22rem 1fr;
  grid-template-rows: auto 1fr auto;
}
header, footer {
  grid-column: 1 / -1;
}
```

▸ Working Demo

27 Grid Row

Contact Form Example 27a

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.



▸ Try it out

HTML

```
<form>
  <div>
    <label> ... </label>
    <input  ...  />
  </div>
  <div>
    <label> ... </label>
```

```
      <input ... />
    </div>
    <div class="message-block">
      <label>...</label>
      <textarea>...</textarea>
    </div>
    <button>...</button>
  </form>
```

Solution

```
.message-block {
  grid-column: 2;
  grid-row-start: 1;
  grid-row-end: 3;
}
button {
  grid-column: 1 / -1;
}
```

▸ Working Demo

We have already learned about the `grid-column` property. Now we're using two new properties `grid-row-start` and `grid-row-end`.

Understanding `grid-row-start` and `grid-row-end`

Concept

The properties `grid-row-start` and `grid-row-end` are again grid item's properties. `grid-row-start` specifies the item's start position and `grid-row-end` specifies the item's end position with respect to row lines. The values are usually positive integers specifying a row line. You can also use a negative integer, in which case it starts counting

the lines in reverse, starting from `-1`.

In the above example, we set the start row line as `1` and end row line as `3`. However you can also use the `span` keyword for either of these values like so.

```
grid-row-start: 1;
grid-row-end: span 2;
```

OR

```
grid-row-start: span 2;
grid-row-end: 3;
```

And similar to `grid-column`, we also have a `grid-row` property that's a shorthand for the start and end properties.

Understanding `grid-row` Concept

The `grid-row` property takes one value or two values separated by forward slash `/`.

One Value

If we specify only one value for this property, that value is `grid-row-start`.

Two Values

The first value is `grid-row-start` and the second one is `grid-row-end`.

So we have three more solutions to Example 27a - Contact form:

```
grid-row: 1 / 3;
```

OR

```
grid-row: 1 / span 2;
```

OR

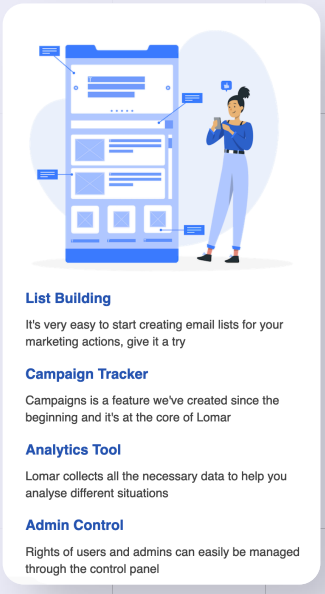
```
grid-row: span 2 / 3;
```

Try them out before you proceed to the next example.

Responsive Services Section Example 27b

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 `grid-column` and `grid-row`.



▸ Try it out

HTML

```
<section>
  <img ... >
  <div id="list1">...</div>
  <div id="list2">...</div>
</section>
```

Solution

Following mobile-first approach, this is how we start - with one single column.

```
section {
  display: grid;
  grid-template-columns: 1fr;
}
```

At a breakpoint of `600px` we change to two columns. We need to place the `#list2` element starting from column line 2, and also make the `img` span across both the rows.

```
@media screen and (min-width: 600px) {
  section {
    grid-template-columns: repeat(2, 1fr);
  }
  #list2 {
    grid-column: 2;
  }
  img {
    grid-row: span 2;
  }
}
```

Then at a breakpoint of `950px` we change to three columns. Here we place the `#list2` starting from column line 3, `#list1` starting from column line 1 and the `img` starting from column line 2. Also, we change back the `img` to span across just one row starting from row line 1.

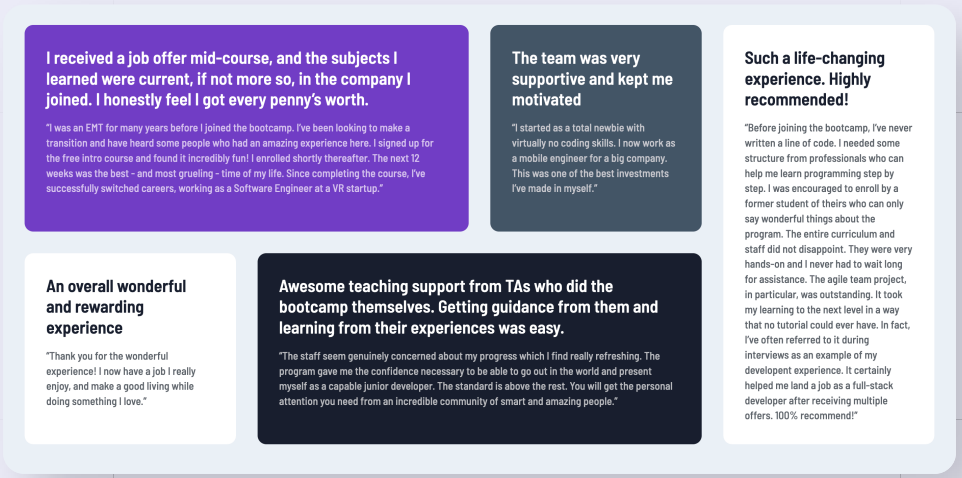
```css
@media screen and (min-width: 950px) {
  section {
    grid-template-columns: repeat(3, 1fr);
  }
  #list1 {
    grid-column: 1;
    text-align: right;
  }
  #list2 {
    grid-column: 3;
  }
  img {
    grid-column: 2;
    grid-row: 1 / span 1;
  }
```

▸ **Working Demo**

Testimonials Grid Section **Example 27c**

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 `1024px` , try and achieve this layout.



Try it out

HTML

```
<section>
  <div class="violet">...</div>
  <div class="gray">...</div>
  <div class="white">...</div>
  <div class="dark">...</div>
  <div class="white-long">...</div>
</section>
```

Sorry for the weird class names

Solution

On mobile, you just have to apply `display: grid` and `gap: 2rem` to `section` and everything just works. Above the breakpoint of `1024px`, you need to create four columns and almost every grid item needs to be positioned using `grid-row` and/or `grid-column`. There are different values that give you the same output. Here's one solution

```
section {
  grid-template-colums: repeat(4, 1fr);
}
.violet {
  grid-column: 1 / span 2; /* Start from column line 1 and span two columns */
}
.white {
  grid-row: 2; /* Start from row line 2 */
}
.dark {
  grid-column: 2 / span 2; /* Start from column line 2 and span two columns */
}
.white-long {
  grid-row: 1 / span 2; /* Start from row line 1 and span two rows */
  grid-column: span 1 / -1; /* Span one column and end at the last column line */
}
```

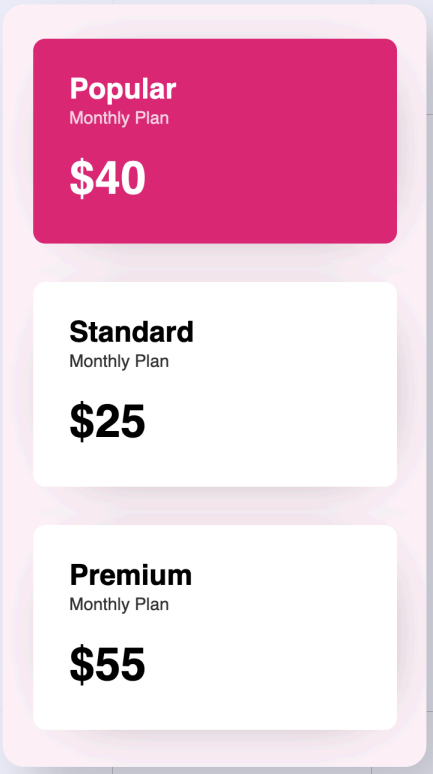
I like to use the grid line number for start / end and use `span` as the other value because it's easy to see how many columns / rows the item is spanning. But you can use the method you're comfortable with.

▸ Working Demo

28 Order

Responsive Pricing Plans Example 28a

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. Just for this example, we will use the desktop first approach.



Try it out

One way to approach this is to change the row lines using `grid-row`. That surely works. But let's look at another solution using the `order` property.

HTML

```
<div class="container">
  <div class="plan">...</div>
  <div class="plan plan-highlight">...</div>
  <div class="plan">...</div>
<div>
```

Solution

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

```
@media screen and (max-width: 600px) {
  .container {
    grid-template-columns: 1fr;
  }
  .plan-highlight {
    order: -1;
  }
}
```

▸ Working Demo

Below the breakpoint of `600px` screen width, we are changing the number of columns to one and changing the order of the highlighted block to `-1`.

Understanding `order` in Grid Concept

The same `order` property 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 in the grid compared to the items with lesser value irrespective of their appearance in the markup.

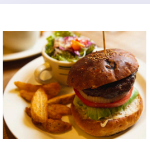
If no `order` is specified, by default the value is `0` for all the elements and they follow the same order as they appear in HTML. Hence when we set `order: -1` to the element in the example above, only that element is removed from the normal flow and placed at the very beginning.

Note : *Just want to give you a heads up that from here on, things start getting a little more complex. So I suggest that you proceed only after you have got a good hold on the concepts covered so far. We will be revisiting a lot of the previous examples making them responsive or figuring out a better solution.*

29 Advanced Grid Template Values

Restaurant Details Example 29a

Let's look an example where we have a restaurant listed for table booking with four columns. The first column is an image and the last column is a button. Let's say we don't want to assign a fixed width to both these columns using `rem` or `px`, but we still want both the column widths to be limited only to the content. And we want the middle two columns to be `auto` widths.



Cheese & Grill Restaurant
St. Marks Street

✓ Outdoor Seating
✓ Private Parking

Book a Table

▸ Try it out

Notice that in the above link, we have set all the columns to `auto` width. That stretches the width of each column to occupy 100% of the container. This has two problems:

1. The image column is wider than the image, creating an unwanted gap between the image and restaurant name
2. The button is wider than the content

Let's see how to solve this.

HTML

```
<div class="container">
  <div><img  ... ></div>
  <div> ... </div>
  <ul> ... </ul>
  <button> ... </button>
</div>
```

Solution

We can use the keyword `max-content` to set the width of the column to be as wide as the content within.

```
.container {
  /* Other rules remain the same */
  grid-template-columns: max-content auto auto max-content;
}
```

▸ Working Demo

This does exactly what we need. Let's understand what `max-content` is after looking at an even better solution.

The Problem

Try resizing the browser to a smaller screen width. You will notice that the button "Book a Table" does not shrink at all even when there's very little space. We are okay if it breaks into two lines when needed, but not stretch when there's more space. In flexbox, this is somewhat like `flex-grow` being `0` while `flex-shrink` being greater than `0`.

A Better Solution

```
.container {
  /* Other rules remain the same */
  grid-template-columns: max-content auto auto fit-content(100%);
}
```

Replace the `grid-template-columns` value with this new one in the above "Working Demo". You will not see any change on large screens. But on smaller screens, the button breaks into two lines. Of course, this is not yet completely responsive, but we'll get to that a little later.

Now, just to experiment, change the `fit-content(100%)` to `min-content` and see what happens.

That column occupies the least width possible by breaking the button into 3 lines. This looks bad on a button, but in some cases, this value might be suitable.

Understanding `max-content`, `min-content` and `fit-content()` Concept

These values can be used for specifying the dimensions of any element, but they're usually used for grid items.

`max-content`

The `max-content` keyword represents the maximum width or height of the content. For text content this means that the content will not wrap at all even if it causes overflows. That's what happened with the button in our previous example.

`min-content`

The `min-content` sizing keyword represents the minimum possible width of the content. For text content this means that the content will wrap everywhere possible making the width equal to the longest word in the content.

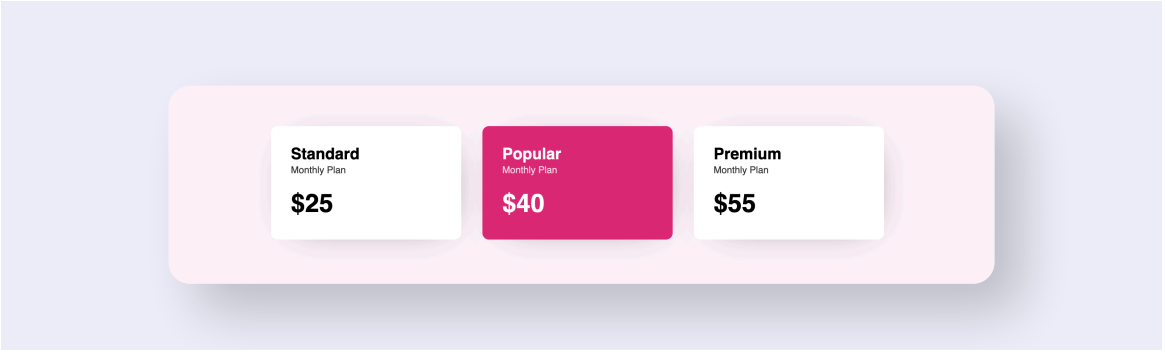
`fit-content()`

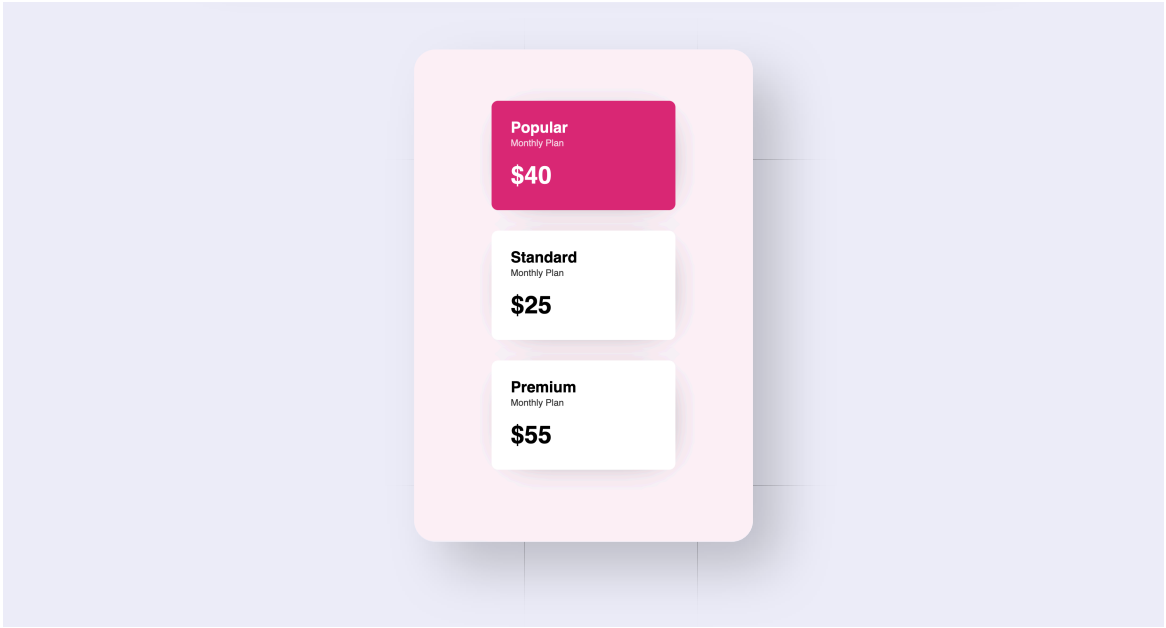
This function accepts a fixed length or percentage as an argument. With respect to grid, this value is similar to `auto`, except when the content is smaller than the available space. In that case, the maximum space is equal to the passed argument.

Example - `fit-content(100%)` : If the space available is less, but the content length is 400px, the element wraps or shrinks to fit into that small space - which is similar to the `auto` value. However if there's more space available, the element does not take up more than 400px *(100% of the content length)*. So, if you use `fit-content(90%)`, the element never takes up more than 360px *(90% of the content length)*.

Pricing Plans with Size Limits **Example 29b**

In the pricing plans example we saw earlier, you might have noticed that the columns stretch full width of the container because we have set `1fr` as the size for each column. 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

```
<div class="container">
  <div class="plan"> ... </div>
  <div class="plan plan-highlight"> ... </div>
  <div class="plan"> ... </div>
<div>
```

Solution

We can add a `max-width` and `min-width` 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:

```css
.container {
  display: grid;
  grid-template-columns: repeat(3, minmax(auto, 18rem));
  justify-content: center; /* To center align all the cards horizontally */
  gap: 2rem;
}

@media screen and (max-width: 600px) {
  .container {
    grid-template-columns: minmax(auto, 18rem);
  }
}
```

▸ Working Demo

We have replaced `1fr` with `minmax(auto, 18rem)`. We can also use `min-content` instead of `auto` - it gives us the same result.

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*.

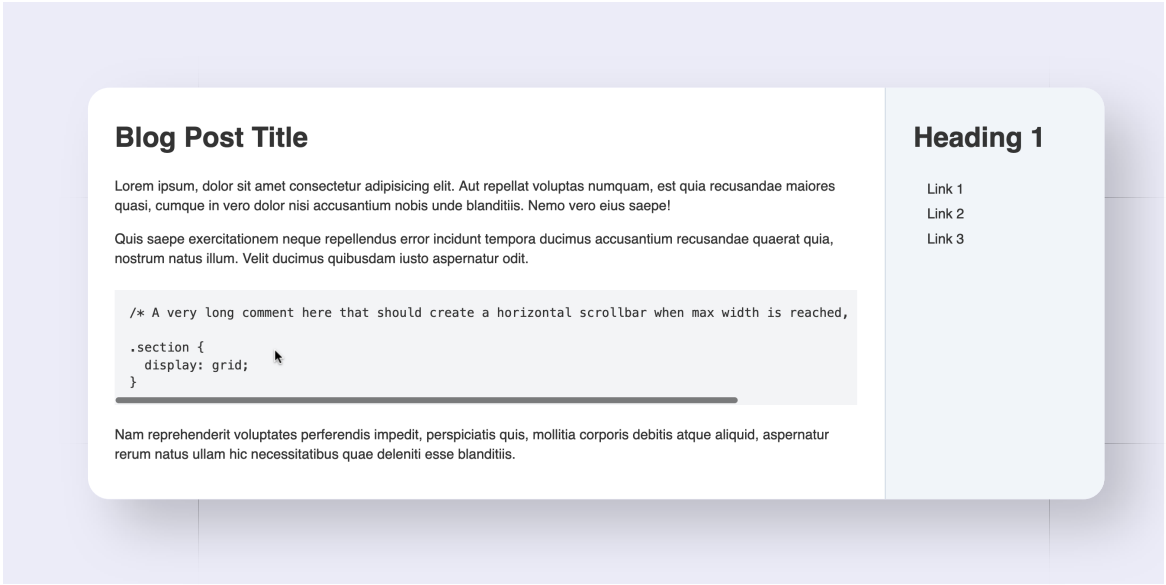
Blog Post Page with Code Snippet Example 29c

Example Inspired from CSS Tricks Article - Preventing a Grid Blowout

We looked at creating a simple page layout with a sidebar using:

```
grid-template-columns: 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: 100%` 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:

HTML

```
<section>
  <div class="main">
    ...
    <pre>...</pre>
    ...
  </div>
  <div class="sidebar">...</div>
</section>
```

Solution

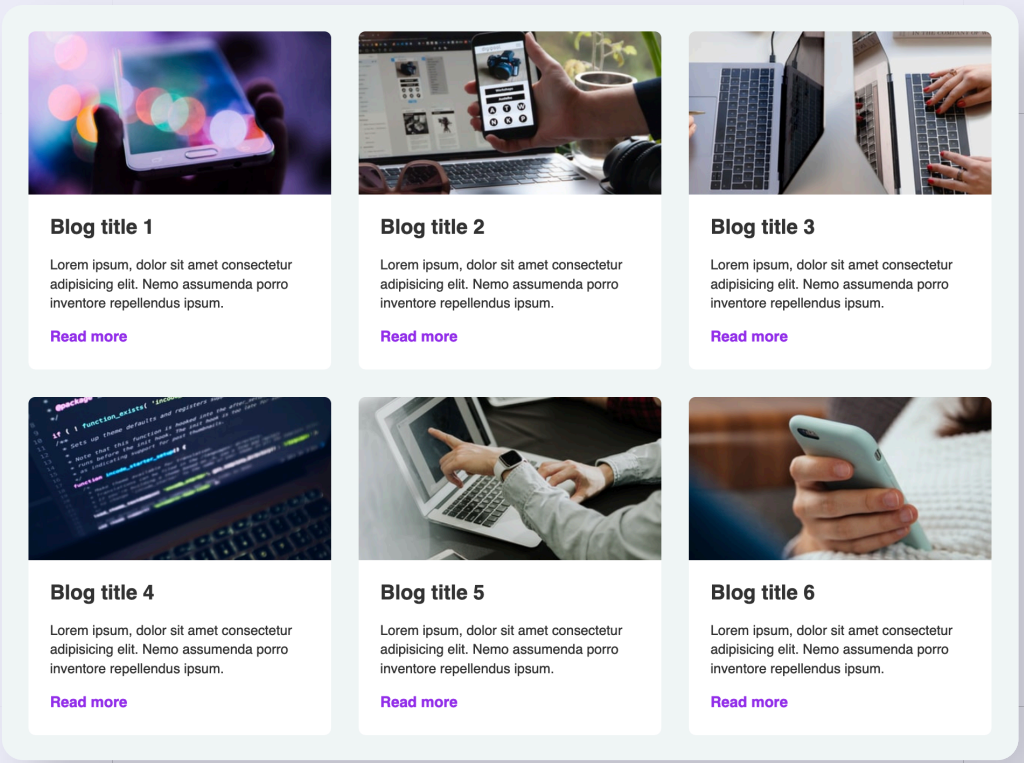
```
section {
  display: grid;
  grid-template-columns: minmax(0, 1fr) 18rem;
}
```

▸ 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`.

Responsive Grid without Media Queries Example 29d

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.



HTML

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

Solution

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

▸ 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

`40rem - 4rem = 36rem`

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

`36rem - 16rem = 20rem`

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:

`20rem - (16rem + 2rem) = 2rem`

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.

```
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?

▸ Try it out

Solution

```
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:

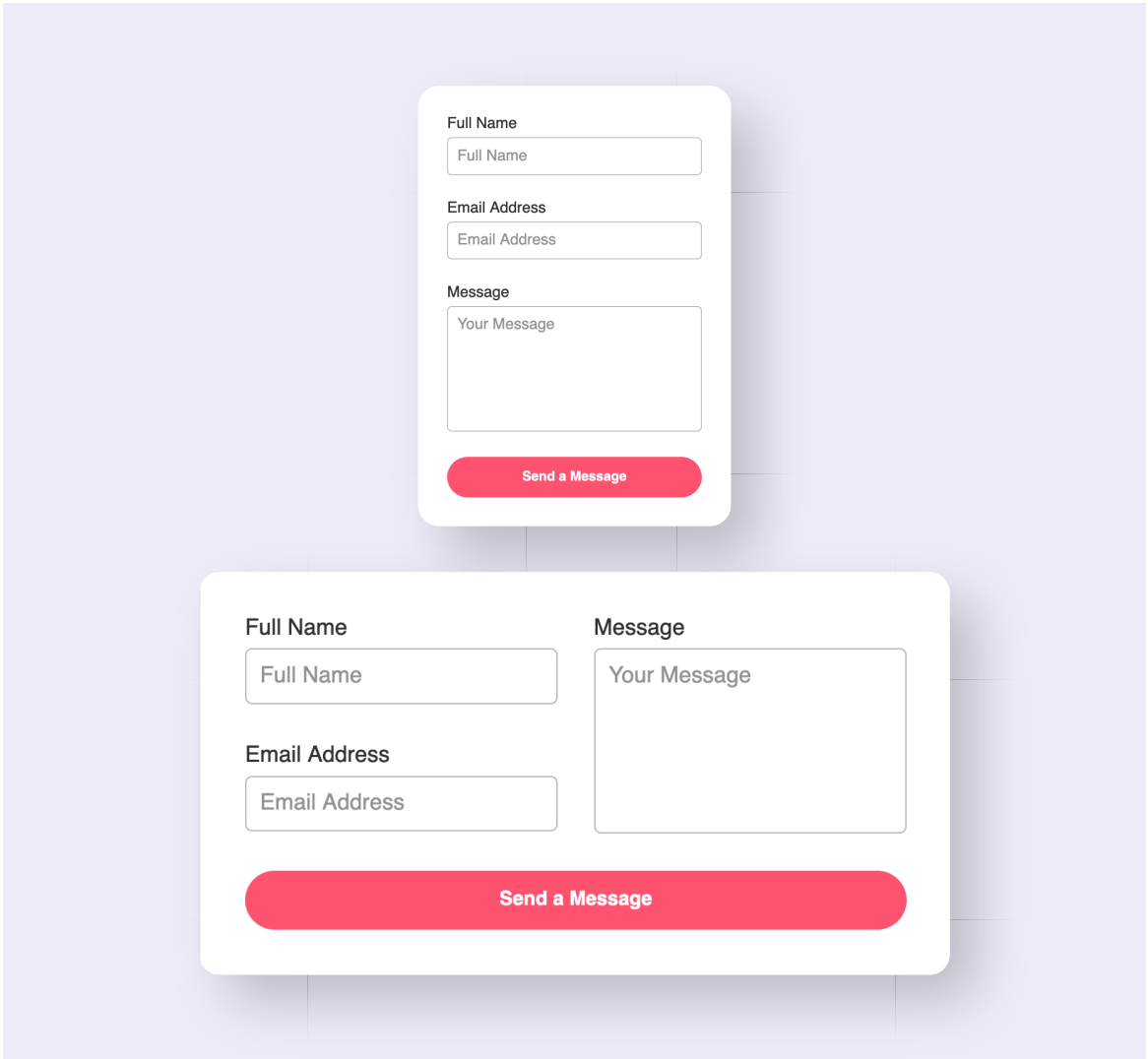
1. Featured Logos in a Grid
2. Responsive Pricing Plans

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

30 Grid Template Areas

Responsive Contact Form Example 30a

Let's revisit the Contact Form Example and make it responsive, this time, using an entirely different approach.



HTML

```
<form>
  <div id="name-block">
    <label> ... </label>
    <input ... />
  </div>
  <div id="email-block">
    <label> ... </label>
    <input ... />
  </div>
  <div id="message-block">
    <label> ... </label>
    <input ... />
  </div>
  <button> ... </button>
</form>
```

Solution

Let me show you the CSS first and then let's understand what's happening. This is for mobile layout:

```
/* Grid Container Styles */
form {
  display: grid;
  grid-template-columns: 1fr;
  grid-template-areas:
    "name"
    "email"
    "message"
    "button";
  gap: 1.6rem;
}
```

```
12
13  /* Grid Items Styles */
14  #name-block {
15    grid-area: name;
16  }
17  #email-block {
18    grid-area: email;
19  }
20  #message-block {
21    grid-area: message;
22  }
23  button {
24    grid-area: button;
25  }
```

Now, to change the layout for screens larger than `530px`, here's what we do:

```
1  @media screen and (min-width: 530px) {
2    form {
3      grid-template-columns: 1fr 1fr;
4      grid-template-areas:
5        "name    message"
6        "email   message"
7        "button button";
8    }
9  }
```

▸ Working Demo

This is a lot to digest at once. We will understand everything step by step.

Understanding `grid-template-areas` Concept

The `grid-template-areas` property is used for assigning each grid cell to a ***named area***. Just looking at this code helps you visualize how it appears on page:

```
grid-template-areas:
  "name   message"
  "email  message"
  "button button";
```

Each word here represents a grid area. You can give any name without spaces. Notice that the `message` word appears twice - one below the other - which makes it span across two rows. Similarly the `button` word appears twice - one next to the other - making it span across two columns.

Note : All these words together have to form a rectangle. So each string must have the same number of words. And the repeated names have to form a rectangle too. An "L" shape or something else is invalid.

Example of invalid values:

```
/* Invalid */
grid-template-areas:
  "name   message"
  "email  message"
  "       button";
/* Invalid */
grid-template-areas:
  "message message"
  "name    message"
  "email   button";
```

If you need a blank cell, you can simple add a dot `.` or a set of dots `...` to represent an empty cell. For example:

```
/* Valid */
grid-template-areas:
  "name   message"
  "email  message"
  " ...    button";
```

Try this out yourself.

In the previous example, for small screens, we used the value:

```
grid-template-areas:
  "name"
  "email"
  "message"
  "button";
```

This is exactly how each cell appears on mobile. And for larger screens, we simply changed the value of `grid-template-columns` to `1fr 1fr` and changed `grid-template-areas` to this:

```
"name   message"
"email  message"
"button button"
```

We don't need to use grid column lines, the span keyword - none of them. This is surely more lines of code, but a lot more readable and easier to change.

Understanding `grid-area` Concept

The property `grid-area` is a shorthand property for grid lines actually. But, in this book, we are going to talk about this property only with respect to named grid areas. So this property must be used to specify a name for **all the grid items** that you want to control. If you miss out on one grid item, that item gets placed automatically where you might not expect.

In the above example, we have four grid items - `#name-block`, `#email-block`, `#message-block` and `button`. We have specified a grid area for each of these items:

```
#name-block {
  grid-area: name;
}
#email-block {
  grid-area: email;
}
#message-block {
  grid-area: message;
}
button {
  grid-area: button;
}
```

Let's try this on one more example for practice.

Responsive Services Section with Grid Template Areas Example 30b

This layout got quite complex while using the row and column lines in Example 27b. Can you try achieving the same using `grid-template-areas`?

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

HTML

```
<section>
  <img ... >
  <div id="list1">...</div>
  <div id="list2">...</div>
</section>
```

Solution

```
img {
  grid-area: img;
}
#list1 {
  grid-area: list1;
}
#list2 {
  grid-area: list2;
}
section {
  display: grid;
  grid-template-columns: 1fr;
  grid-template-areas:
    "img"
    "list1"
    "list2";
  column-gap: 2rem;
}
@media screen and (min-width: 600px) {
  section {
    grid-template-columns: repeat(2, 1fr);
    grid-template-areas:
      "img list1"
      "img list2";
  }
}
```

```
@media screen and (min-width: 950px) {
  section {
    grid-template-columns: repeat(3, 1fr);
    grid-template-areas: "list1 img list2";
  }
}
```

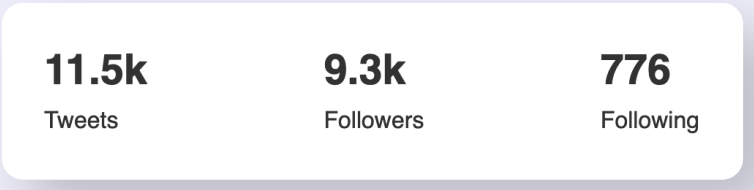
▸ Working Demo

If you want one more exercise for practice, try out this responsive layout on MDN Docs.

31 Grid Auto Flow

Analytics Section Example 31a

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

```
<section>
  <span>11.5k</span>
  <p>Tweets</p>
  <span> ... </span>
  <p> ... </p>
  <span> ... </span>
  <p> ... </p>
</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:

```
section {
  display: grid;
  grid-template-rows: auto auto; /* We are creating two rows */
  grid-auto-flow: column; /* Changing the flow */
  justify-content: space-between; /* Spacing the columns wide apart */
  row-gap: 0.5rem;
}
```

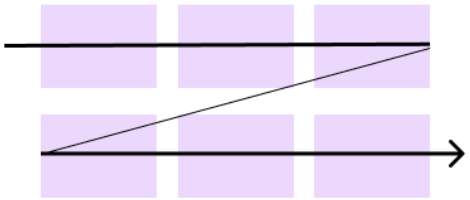
▸ Working Demo

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

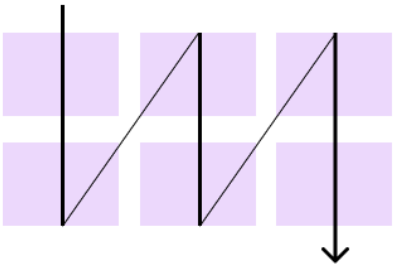
Understanding `grid-auto-flow` Concept

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

The other commonly used value is `column` which changes the flow to fill the first column and then keep adding more columns as we saw in our previous example.



grid-auto-flow: row



grid-auto-flow: column

32 Grid Auto Rows

Gallery Example 32a

Let's assume we need a responsive gallery page with alternate rows to be double the height of the other rows. We can use the `auto-fit` keyword in `grid-template-columns` to make the whole page responsive. But we cannot use `grid-template-rows` to specify the heights of rows because we don't know how many images will be fetched. These rows are being created implicitly. So how do we specify the height of rows?

Try it out

HTML

```
<section>
  <img ... >
  <!-- 'n' number of images -->
</section>
```

Solution

```
section {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(20rem, 1fr));
  grid-auto-rows: 1fr 2fr;
  gap: 1rem;
}
```

▸ Working Demo

We are using the property `grid-auto-rows` with a value that repeats itself when additional rows are created.

Understanding `grid-auto-rows` Concept

The **`grid-auto-rows`** property specifies the size of an implicitly created grid rows or a pattern of rows, like in our previous example. The values could be in fixed units like `px`, `rem` or in percentages `%` or keywords like min-content, max-content or also functions like fit-content() and max-content().

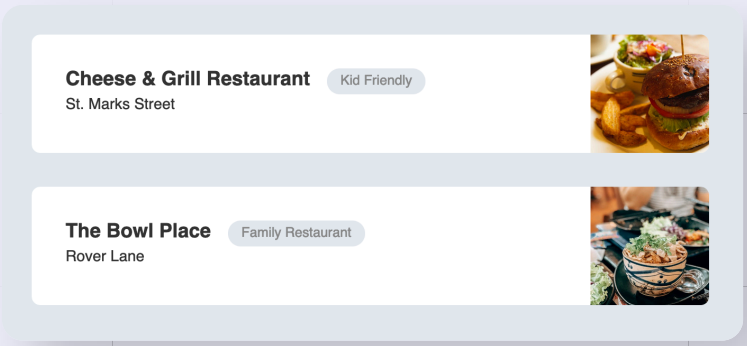
When a single value is provided, all the rows are sized with that value. And when multiple values are provided, like `1fr 2fr`, it creates a pattern that repeats itself.

There's a similar property **`grid-auto-columns`** to specify the size of implicitly created columns when you don't specify the column size using `grid-template-columns`. One possibility is when you set `grid-auto-flow` as `column`.

33 Justify Self & Align Self

Restaurant Cards with Labels Example 33a

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

HTML

```
<div class="container">
  <div class="info"> ... </div>
  <span class="label"> ... </span>
  <img  ... >
</div>
```

Solution

```
.container {
  display: grid;
  grid-template-columns: auto auto 1fr;
}
.label {
  align-self: start; /* To align the label to the top */
}
img {
  justify-self: end; /* To push the image to the far right of the 1fr column
}
```

▸ 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 `align-self` on the image to push the image to the right.

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

Let's learn more about these two new properties.

Understanding `justify-self` and `align-self`

Concept

The property `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).

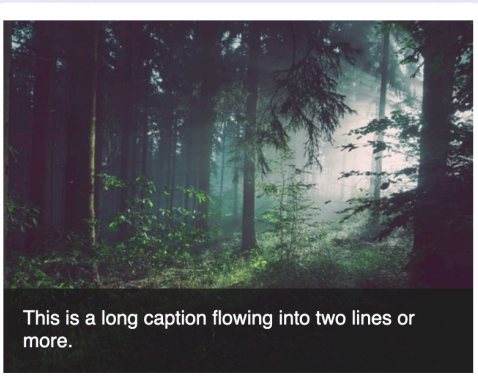
The property `align-self` is also used on a grid item. When the content of the item is shorter than the height of the row, we can use this property to control the alignment along the *block axis* (vertical direction).

The values are similar to that of `justify-items` and `align-items`. Some of the commonly used values are `start`, `end`, `center` and `stretch`.

In the above example, `align-self` is used to align the label to the top of the row with the value `start`. And the property `justify-self` is used to align the image to the right using the value `end`.

Caption at the Bottom of Image Example 33b

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

HTML

```
<figure>
  <img ... >
  <figcaption> ... </figcaption>
</figure>
```

Solution

```css
figure {
  display: grid;
  grid-template-areas: "figure";
}
img {
  grid-area: figure;
}
figcaption {
  grid-area: figure;
  align-self: end;
}
```

▸ Working Demo

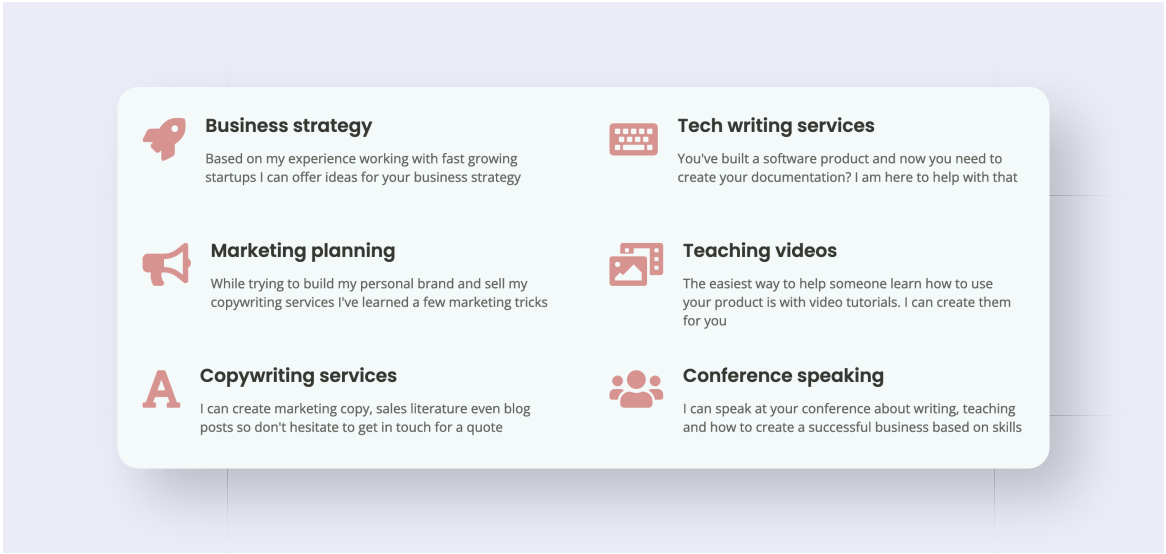
We are using `grid-template-areas` to create a single row and single column with an area name "figure". Then we specify the same grid area for both `img` and `figcaption` which makes it overlap! Yes, it's possible to create overlapping elements with grid. Now both the elements occupy the same grid cell. So, we need to use `align-self: end` to push down the caption.

Comprehensive Examples for Grid & Flexbox

Services Section Example 34a

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

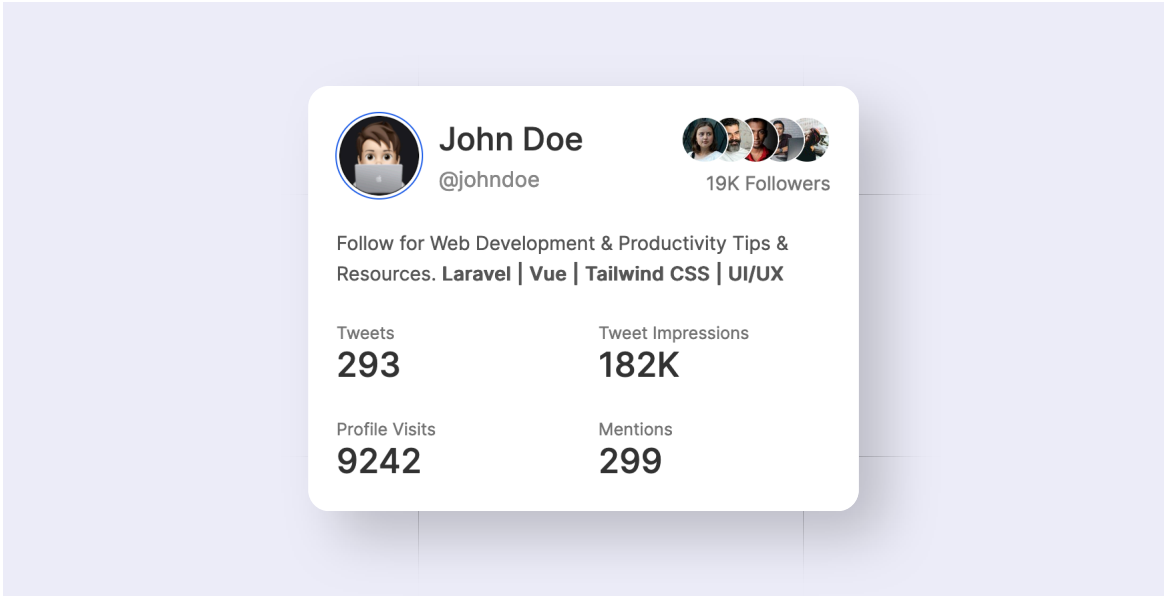
HINT : Use the properties `grid-template-rows` and `grid-auto-flow` for the grid. And make each grid item a flex container.

▸ Working Demo

Twitter Monthly Summary Card Example 34b

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 `align-items` and auto margins within flexbox. Use `row-reverse` direction for the followers' images. For the grid, simply use `grid-template-columns` and `gap`.

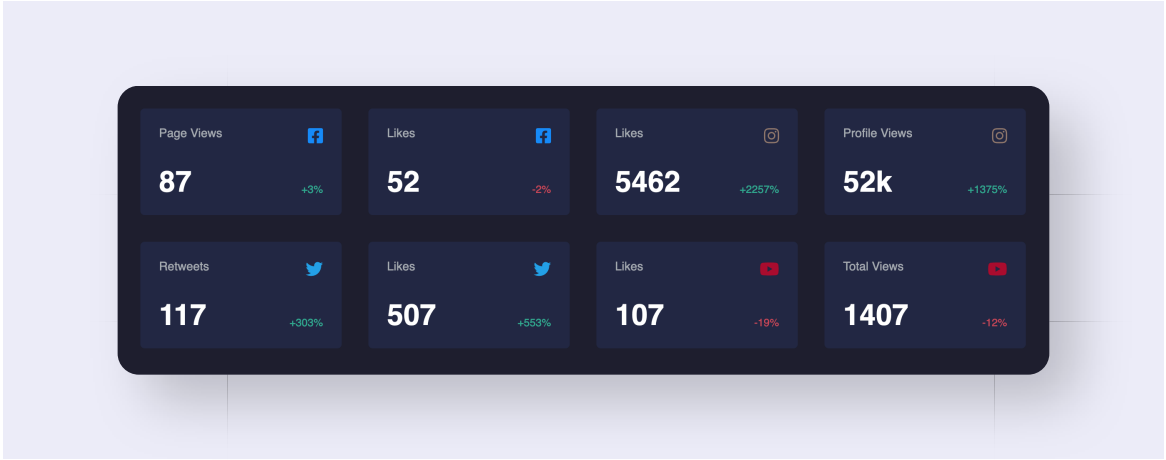
▸ Working Demo

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

Social Media Dashboard Example 34c

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 properties `grid-template-columns`, `grid-auto-rows`, `justify-content`, `align-content`, `justify-self`, `align-self` and `gap`.

▸ Working Demo

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. And watch out for updates to the book. Thank you 🙏