CSS Flex & Grid

Complete Guide with Real World Examples and Code Snippets

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."

Table of Contents

Introduction

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

What not to expect

1. I will not be going through the concepts in the order in which they are usually covered in other tutorials.
2. 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.

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`, `color`, `background`, `border`, `position`, `float` and concepts of viewport, responsive web design and media queries.

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 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 couple of 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 : Now we will get to explaining the concept and understand 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 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***

STEP 7 : You will see a summary at the end of multiple concepts to reinforce all that you just learnt.

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. There are some generic styles added to CSS within the comments block `/* Generic Styles */`. You can usually ignore the styles within this block.

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. 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 adjust widths based on the length of each quote and occupy the same height. These quotes are randomly picked. You don't know how long or short each one is, so you cannot specifiy widths 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

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

Solution

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

```css
/* 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.

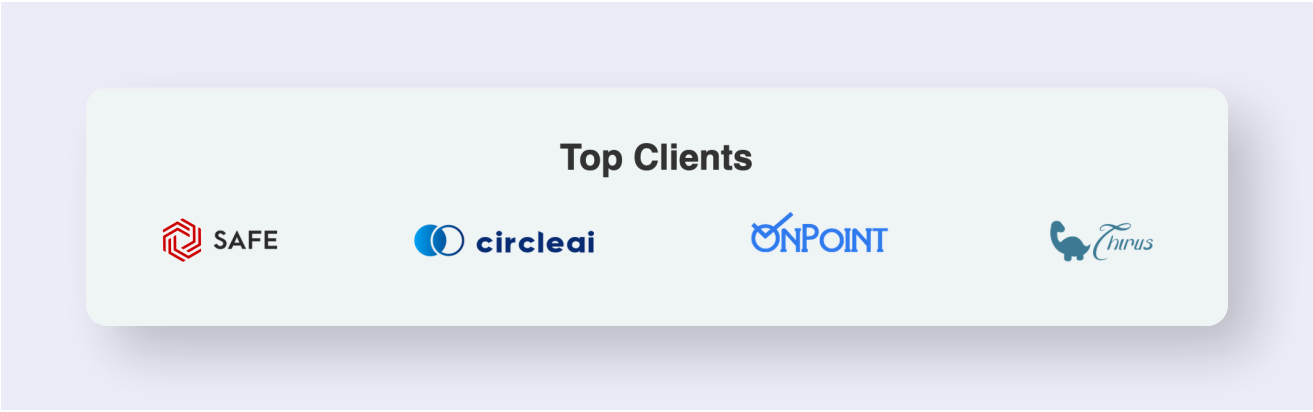
Note: Only the immediate child elements of the container become flex items. Children of flex items are not affected.

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

Logos Spaced Out Example 2a

Let's say you need to display a few logos of your clients and you want them to space out fully with the first logo on the extreme left, last logo on the extreme right and the middle ones spaced out evenly. These logos have different dimensions. 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 acheive this with flexbox.

HTML

```
<div class="wrapper">
  <h2>Top Clients</h2>
  <div class="logos">
    <img src="..." />
    <img src="..." />
    <img src="..." />
    <img src="..." />
  </div>
</div>
```

Solution

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

```css
.logos {
  /* 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 `diplay: flex` to an element, we saw that the children get placed next to each other in one single row. This is a default behavoiur. However, we can place them all one below the other in a single column instead. We will get to that a little later.

`justify-content` is the property we can use to control the 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

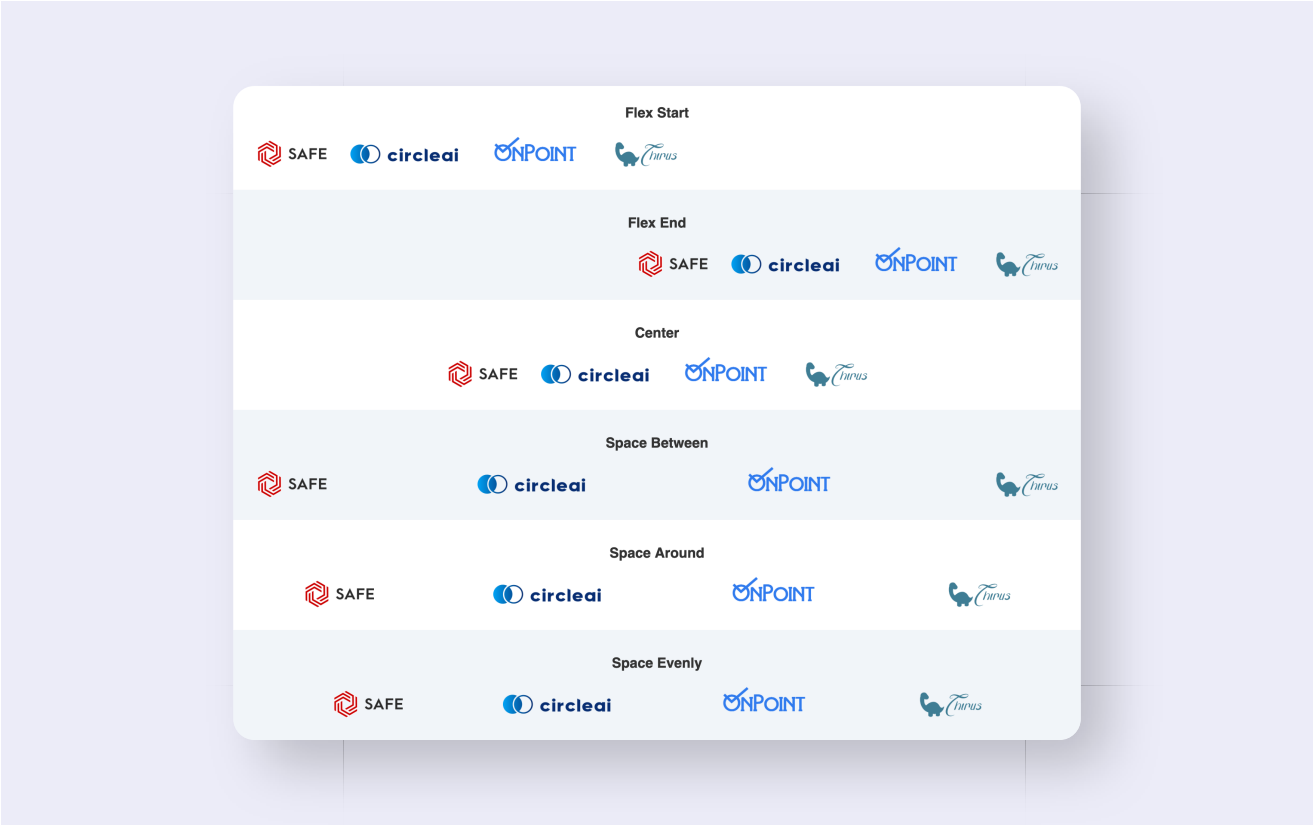
`space-around`

Space 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

You can see the difference between these values below:



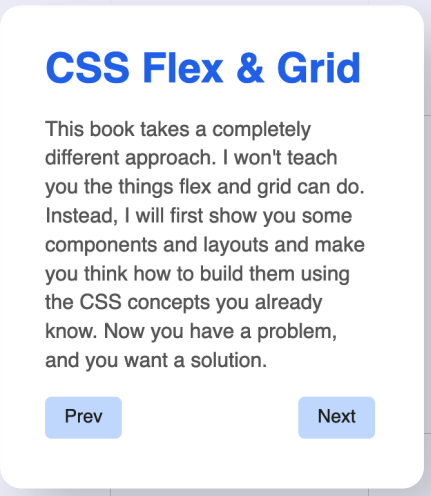
Open the working demo, resize the output panel and see how the items move.

▸ 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.

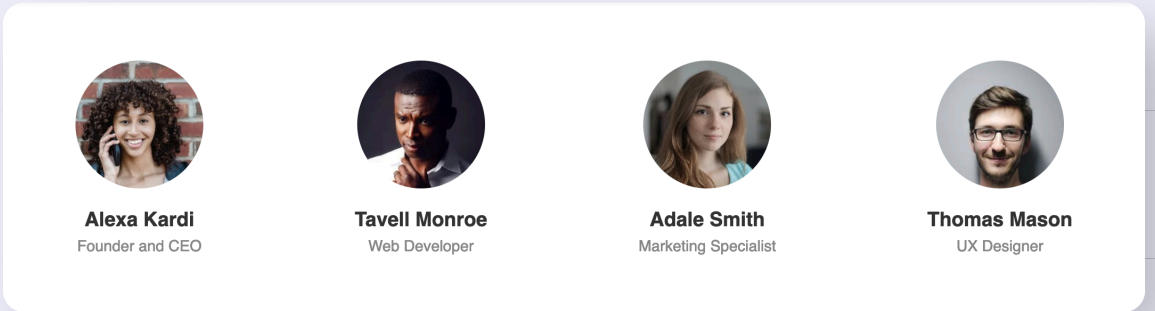


Now that you have seen one example, try this out on your own and cross check with the working demo.

▸ 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.



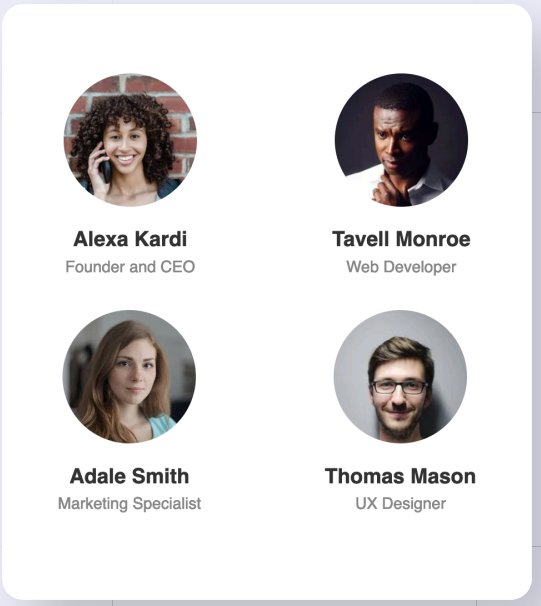
Try it out yourself in the CodePen link below. HINT: Add styles to the `.container` selector at the end of the CSS code.

▸ 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 notice a horizontal scrollbar. 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;
}
```

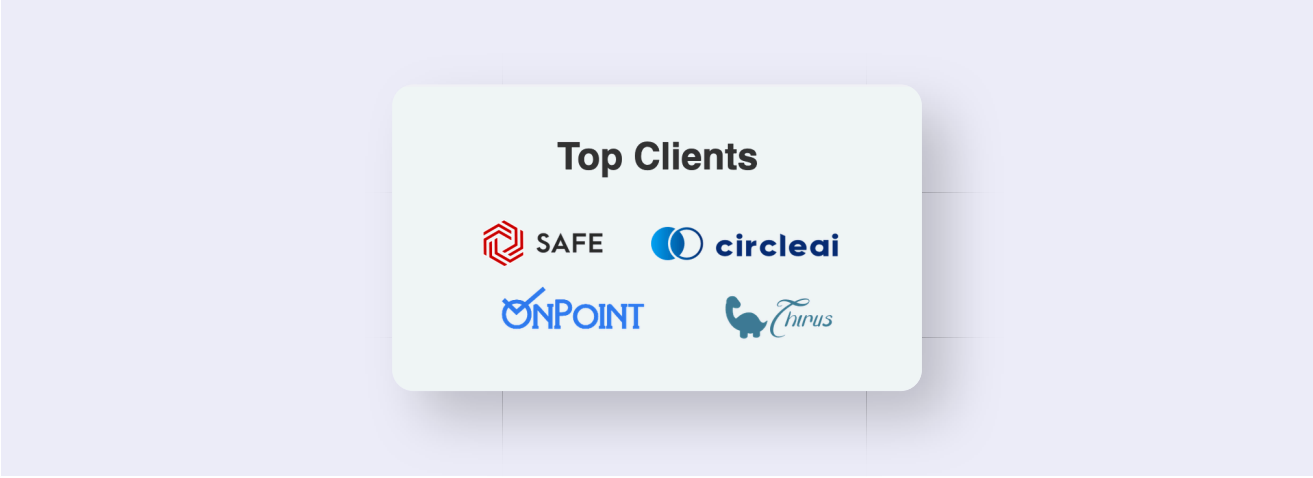
▸ Working Demo

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 `no-wrap` 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.

You can try the same with the Logos example too. And for better appearance, set the `justify-content` property to `space-around`

Logos Wrapped Example 3b



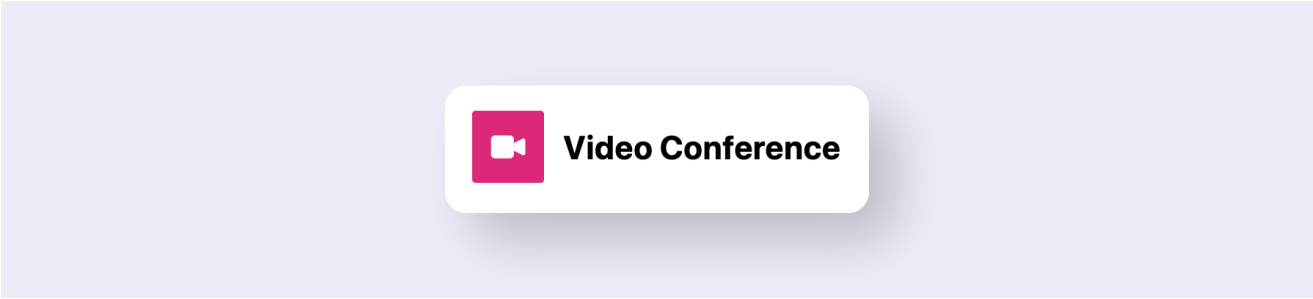
▸ Working Demo

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

4 Align Items

Icon and Text Example 4a

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.

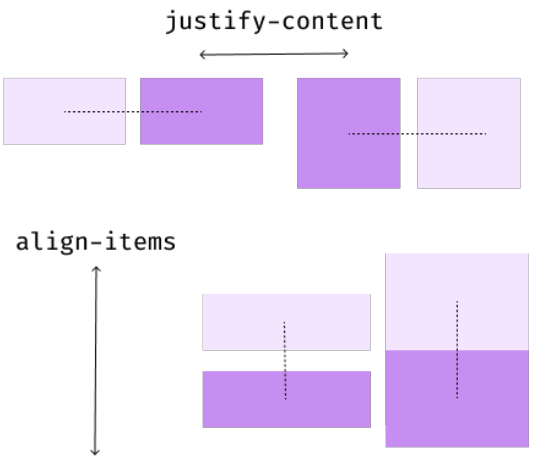
```
.icon-wrap {
  /* Existing styles */
  display: flex;
  align-items: center;
}
```

▸ 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. While `align-items` property allows to control the alignment in its perpendular 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 placed at the center of the container

`flex-start`

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

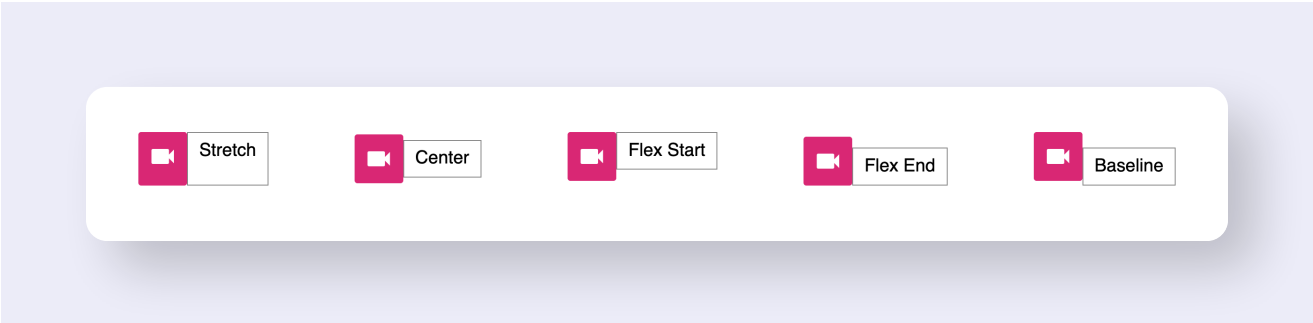
`flex-end`

All items are spaced at 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

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

to

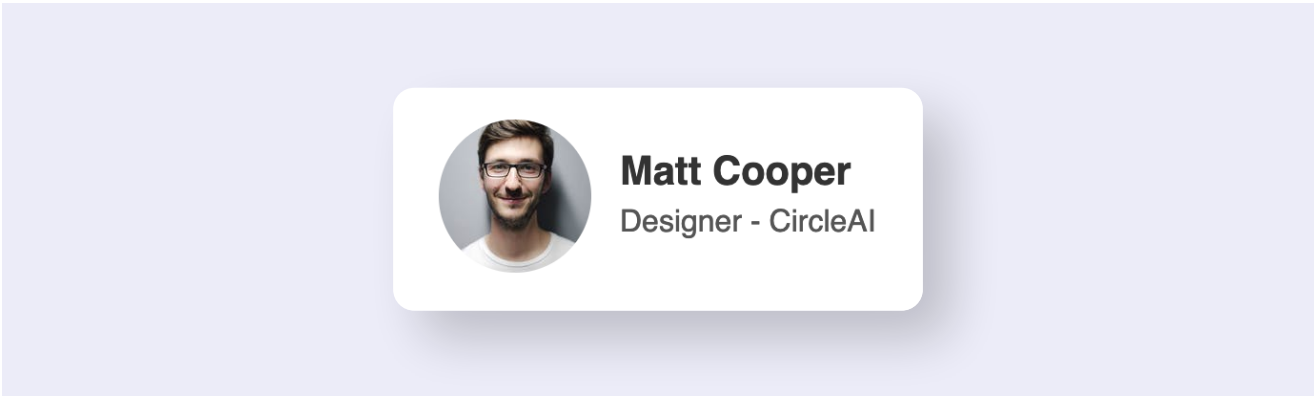
```
<span class="icon material-icons">V</span>
```

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

So many times we need to design a small component with a small image and a couple of lines next to it. The `align-items` property with value `center` is very useful for such requirements

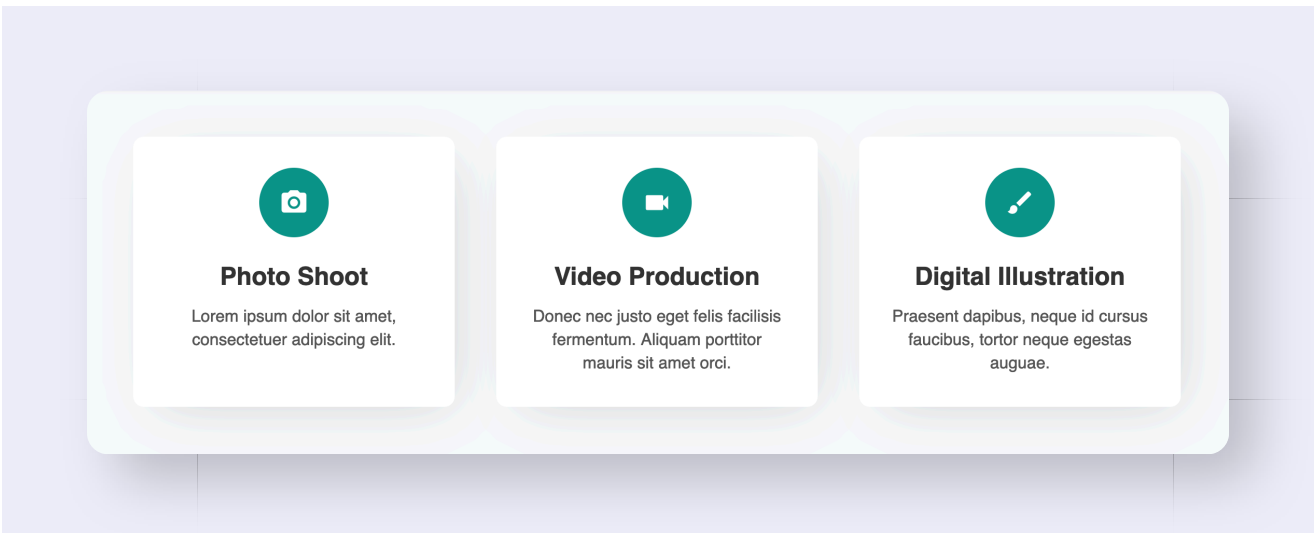


Try doing this yourself before looking at the working code

▸ 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.

Center Anything **Example 4d**

This is something you will always encounter. You want to center an element within its parent, but there's no straightforward way to center an element 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.