The ultimate CSS battle: Grid vs Flexbox

Learn how they differ, and when you should use one over the other.

CSS Flexbox has become extremely popular amongst front-end developers the last couple of years. This isn't surprising, as it has made it a lot easier for us to create dynamic layouts and align content within containers.

However, there's a new kid in town called CSS Grid, and it's got a lot of the same capabilities as Flexbox. In come cases it's better than Flexbox, while in other cases it's not.

This seems to be a source of confusion for developers. So in this article compare the two modules, both a micro and macro level.

If you want to learn the two modules properly, check out my free courses CSS Grid and CSS Flexbox.

Now let's get started!

One dimension vs two dimensions

If you are to take one lesson from this article, let it be this one:

Flexbox is made for one dimensional layouts and Grid is made for two dimensional layouts.

Yup right

This means that if you're laying out items in one direction (for example three buttons inside a header), then you should use Flexbox:

It'll will give you more flexibility than CSS Grid. It'll also be easier to maintain and require less code.

However if you're going to create an entire layout in two dimensions—with both rows *and* columns—then you should use CSS Grid:

In this case CSS Grid will give you more flexibility, make your markup simpler and the code will be easier to maintain.

Now you can of course combine the two. In the example above it would be perfect to use Grid for the page layout, and then Flexbox to align the content inside the header. This'll give you the best of both worlds. And I'll show you exactly how to do at the end of this article.

Content-first vs layout-first

Another core difference between the two is that Flexbox takes basis in the **content** while Grid takes basis in the **layout**. This might seem abstract, so let's look at a specific example, as that makes it easier to understand.

We'll use the header from the previous paragraph. Here's the HTML for it:

```
<header>
  <div>Home</div>
  <div>Search</div>
  <div>Logout</div>
</header>
```

Before we turned it into a Flexbox layout these div's would have been stacked on top of each other like this:

I've added a little bit of basic styling, which as nothing to do with Flexbox or Grid, so I'm leaving that out.

Flexbox header

However, when we give it a `display: flex;` the items will be places nicely on a line.

```
header {
  display: flex;
}
```

To move the *logout* button to the far right side, we'll simply target that element and give it a margin:

```
header > div:nth-child(3) {
  margin-left: auto;
}
```

Which results in the following:

What I want you to notice here is that we leave it up to the items themselves to decide how they're placed. We didn't have to pre-define anything else than `display: flex;` initially.

This is at the core of the difference between Flexbox and Grid, and it will be more apparent as we recreate this header using Grid.

Even though CSS Grid isn't build to create one-dimensional headers, it's still a good exercise to do it in this article, as it teaches us about the core differences between Flexbox and Grid.

Grid header

We can create our header in several different ways using CSS Grid. I'm going to go with a pretty straight forward one, where our grid has ten columns, each being one fraction unit wide.

```
header {
  display: grid;
  grid-template-columns: repeat(10, 1fr);
}
```

It'll look identical to the Flexbox solution.

However, we can peak under the hood to see what's different. We'll use the Chrome inspector to inspect the column lines:

The key difference with this approach is that we had to define the columns—the layout—first. We start with defining the width of the columns, and then we place the content in the available grid cells.

This approach forced us to take a stance on how many columns we wanted to split our header into.

Unless we change the grid, we're stuck with ten columns. A limitation we wouldn't have had to deal with in Flexbox.

In order to change the *logout* to the far right hand side, we'll place it in the tenth column, like this:

```
header > div:nth-child(3) {
  grid-column: 10;
}
```

Here's how that looks when we're inspecting the grid:

We couldn't simply have given it a `margin-left: auto;` because the *logout* button had already been placed in a specific cell in the layout, in the third column. To move it, we had to find another grid cell for it.

Combining the two

Now let's look at how to use both in combination, merging our header into our website layout. We'll start by building the website layout.

Here's the markup:

```
<div class="container">
  <header>HEADER</header>
  <aside>MENU</aside>
  <main>CONTENT</main>
  <footer>FOOTER</footer>
</div>
```

Here's the CSS:

```
.container {
  display: grid;
  grid-template-columns: repeat(12, 1fr);
  grid-template-rows: 50px 350px 50px;
}
```

We'll place the items on the grid like this:

```
header {
  grid-column: span 12;
}

aside {
  grid-column: span 2;
}

main {
  grid-column: span 10;
}

footer {
  grid-column: span 12;
}
```

Now we'll simply add the header. We'll turn the header—which is an item in our CSS Grid—into a Flexbox container.

```
header {
  display: flex;
}
```

Now we can set the logout button to the right:

```
header > div:nth-child(3) {
  margin-left: auto;
}
```

And there we have a perfectly fine layout which uses the best from both Grid and Flexbox. Here's how the two containers look:

ontainer **Flexbox cont**

So now you should have a strong understanding of the general and specific differences between Flexbox and Grid, and know how to use them together.

Browser support

Before we end, I also need to mention browser support. At the time of writing this article, 77% of global website traffic supports CSS Grid, and it's climbing.

Click the image below to see a preview of the course.