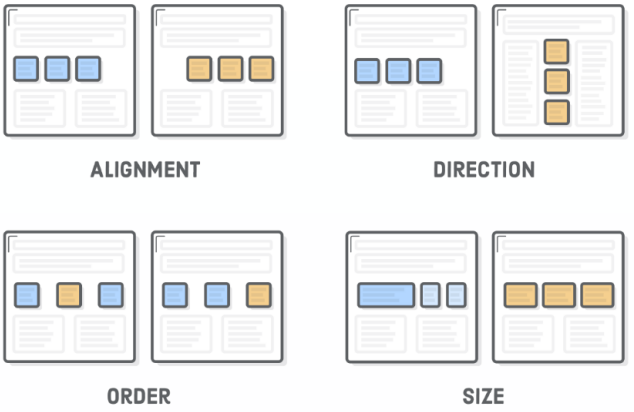
FLEXBOX

Nº 8. OF HTML & CSS IS HARD

A friendly tutorial for modern CSS layouts

The "Flexible Box" or "Flexbox" layout mode offers an alternative to Floats for defining the overall appearance of a web page. Whereas floats only let us horizontally position our boxes, flexbox gives us *complete* control over the alignment, direction, order, and size of our boxes.



The web is currently undergoing a major transition, so a little discussion around the state of the industry is warranted. For the last decade or so, floats were the sole option for laying out a complex web page. As a result, they're well supported even in legacy browsers, and developers have used them to build millions of web pages. This means you'll inevitably run into floats during your web development career (so the previous chapter wasn't a total waste).

However, floats were originally intended for the magazine-style layouts that we covered in Floats for Content. Despite what we saw last chapter, the kinds of layouts you can create with floats are actually somewhat limited. Even a simple sidebar layout is, technically speaking, a little bit of a hack. Flexbox was invented to break out of these limitations.

We're finally at a point where browser support has hit critical mass and developers can start building full websites with flexbox. Our recommendation is to use flexbox to lay out your web pages as much as possible, reserving floats for when you need text to flow *around* a box (i.e., a magazine-style layout) or when you need to support legacy web browsers.

In this chapter, we'll explore the entire flexbox layout model step by step. You should walk away comfortable building virtually any layout a web designer could ever give you.

SETUP

The example for this chapter is relatively simple, but it clearly demonstrates all of the important flexbox properties. We'll wind up with something that looks like this:

For starters, we need an empty HTML document that contains nothing but looks like this:

For starters, we need an empty HTML document that contains nothing but a menu bar. Make a new Atom project called `flexbox` to house all the example files for this chapter. Then, create `flexbox.html` and add the following markup:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Some Web Page</title>
    <link rel='stylesheet' href='styles.css'/>
  </head>
  <body>
    <div class='menu-container'>
      <div class='menu'>
        <div class='date'>Aug 14, 2016</div>
        <div class='signup'>Sign Up</div>
        <div class='login'>Login</div>
      </div>
    </div>
  </body>
</html>
```

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Some Web Page</title>
    <link rel='stylesheet' href='styles.css'/>
  </head>
  <body>
    <div class='menu-container'>
      <div class='menu'>
        <div class='date'>Aug 14, 2016</div>
        <div class='signup'>Sign Up</div>
        <div class='login'>Login</div>
      </div>
    </div>
  </body>
</html>
```

Next, we need to create the corresponding `styles.css` stylesheet. This won't look like much: just a full-width blue menu bar with a white-bordered box in it. Note that we'll be using flexbox instead of our traditional auto-margin technique to center the menu.

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

.menu-container {
  color: #fff;
  background-color: #5995DA;  /* Blue */
  padding: 20px 0;
}

.menu {
  border: 1px solid #fff;  /* For debugging */
  width: 900px;
}
```

margin technique to center the menu.

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

.menu-container {
  color: #fff;
  background-color: #5995DA;  /* Blue */
  padding: 20px 0;
}

.menu {
  border: 1px solid #fff;  /* For debugging */
  width: 900px;
}
```

Finally, download some images for use by our example web page. Unzip them into the `flexbox` project, keeping the parent `images` directory. Your margin technique to center the menu.

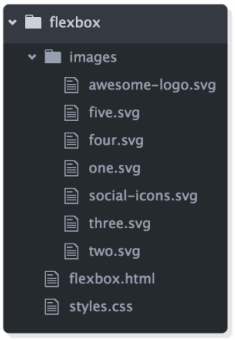
```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

.menu-container {
  color: #fff;
  background-color: #5995DA;  /* Blue */
  padding: 20px 0;
```

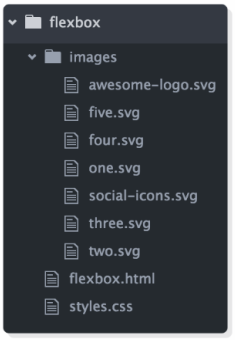
```
}

.menu {
  border: 1px solid #fff;  /* For debugging */
  width: 900px;
}
```

Finally, download some images for use by our example web page. Unzip them into the `flexbox` project, keeping the parent `images` directory. Your project should look like this before moving on:



Finally, download some images for use by our example web page. Unzip them into the `flexbox` project, keeping the parent `images` directory. Your project should look like this before moving on:



FLEXBOX OVERVIEW

Flexbox uses two types of boxes that we've never seen before: "flex containers" and "flex items". The job of a flex container is to group a bunch of flex items together and define how they're positioned.

FLEXBOX OVERVIEW

Flexbox uses two types of boxes that we've never seen before: "flex containers" and "flex items". The job of a flex container is to group a bunch of flex items together and define how they're positioned.

"FLEX CONTAINER" "FLEX ITEMS"

Every HTML element that's a direct child of a flex container is an "item". Flex items can be manipulated individually, but for the most part, it's up to the container to determine their layout. The main purpose of flex items are to let their container know how many things it needs to position.

As with float-based layouts, defining complex web pages with flexbox is all about nesting boxes. You align a bunch of flex items inside a container, and, in turn, those items can serve as flex containers for their own items. As you work through the examples in this chapter, remember that the fundamental task of laying out a page hasn't changed: we're still just moving a bunch of nested boxes around.

Every HTML element that's a direct child of a flex container is an "item". Flex items can be manipulated individually, but for the most part, it's up to the container to determine their layout. The main purpose of flex items are to let their container know how many things it needs to position.

As with float-based layouts, defining complex web pages with flexbox is all about nesting boxes. You align a bunch of flex items inside a container, and, in turn, those items can serve as flex containers for their own items. As you work through the examples in this chapter, remember that the fundamental task of laying out a page hasn't changed: we're still just moving a bunch of nested boxes around.

FLEX CONTAINERS

The first step in using flexbox is to turn one of our HTML elements into a flex container. We do this with the `display` property, which should be

As with float-based layouts, defining complex web pages with flexbox is all about nesting boxes. You align a bunch of flex items inside a container, and, in turn, those items can serve as flex containers for their own items. As you work through the examples in this chapter, remember that the fundamental task of laying out a page hasn't changed: we're still just moving a bunch of nested boxes around.

FLEX CONTAINERS

The first step in using flexbox is to turn one of our HTML elements into a flex container. We do this with the `display` property, which should be familiar from the CSS Box Model chapter. By giving it a value of `flex`, we're telling the browser that everything in the box should be rendered with flexbox instead of the default box model.

Add the following line to our `.menu-container` rule to turn it into a flex

container:

```
.menu-container {
  /* ... */
  display: flex;
}
```

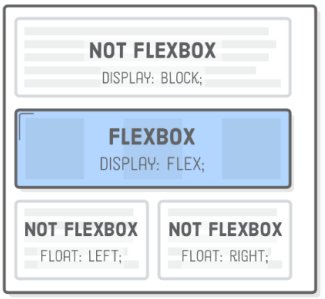
FLEX CONTAINERS

The first step in using flexbox is to turn one of our HTML elements into a flex container. We do this with the `display` property, which should be familiar from the CSS Box Model chapter. By giving it a value of `flex`, we're telling the browser that everything in the box should be rendered with flexbox instead of the default box model.

Add the following line to our `.menu-container` rule to turn it into a flex container:

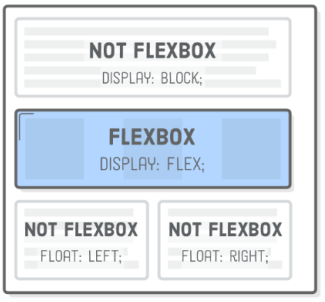
```
.menu-container {
  /* ... */
  display: flex;
}
```

This *enables* the flexbox layout mode—without it, the browser would ignore all the flexbox properties that we're about to introduce. Explicitly defining flex containers means that you can mix and match flexbox with other layout models (e.g., floats and everything we're going to learn in Advanced Positioning).

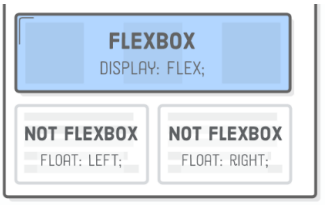


Great! We have a flex container with one flex item in it. However, our page will look exactly like it did before because we haven't told the container how to display its item.

models (e.g., floats and everything we're going to learn in Advanced Positioning).



Great! We have a flex container with one flex item in it. However, our page will look exactly like it did before because we haven't told the container how to display its item.



Great! We have a flex container with one flex item in it. However, our page will look exactly like it did before because we haven't told the container how to display its item.

ALIGNING A FLEX ITEM

After you've got a flex container, your next job is to define the horizontal alignment of its items. That's what the `justify-content` property is for. We can use it to center our `.menu`, like so:

```
.menu-container {
  /* ... */
  display: flex;
  justify-content: center;    /* Add this */
}
```

ALIGNING A FLEX ITEM

After you've got a flex container, your next job is to define the horizontal alignment of its items. That's what the `justify-content` property is for. We can use it to center our `.menu`, like so:

```
.menu-container {
  /* ... */
  display: flex;
  justify-content: center;    /* Add this */
}
```

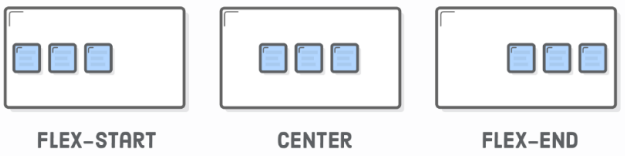
This has the same effect as adding a `margin: 0 auto` declaration to the `.menu` element. But, notice how we did this by adding a property to the *parent* element (the flex container) instead of directly to the element we wanted to center (the flex item). Manipulating items through their containers like this is a common theme in flexbox, and it's a bit of a

After you've got a flex container, your next job is to define the horizontal alignment of its items. That's what the `justify-content` property is for. We can use it to center our `.menu`, like so:

```
.menu-container {
  /* ... */
  display: flex;
  justify-content: center;    /* Add this */
}
```

This has the same effect as adding a `margin: 0 auto` declaration to the `.menu` element. But, notice how we did this by adding a property to the *parent* element (the flex container) instead of directly to the element we wanted to center (the flex item). Manipulating items through their containers like this is a common theme in flexbox, and it's a bit of a

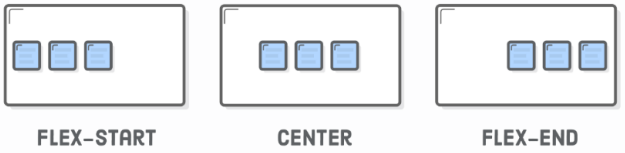
divergence from how we've been positioning boxes thus far.



Other values for `justify-content` are shown below:

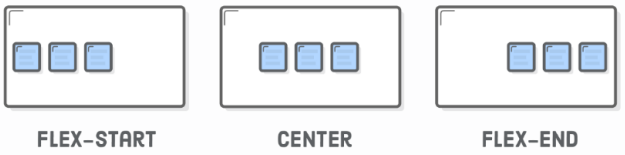
- `center`

This has the same effect as adding a `margin: 0 auto` declaration to the `.menu` element. But, notice how we did this by adding a property to the *parent* element (the flex container) instead of directly to the element we wanted to center (the flex item). Manipulating items through their containers like this is a common theme in flexbox, and it's a bit of a divergence from how we've been positioning boxes thus far.



Other values for `justify-content` are shown below:

- `center`
- `flex-start`
- `flex-end`
- `space-around`
- `space-between`



Other values for `justify-content` are shown below:

- `center`
- `flex-start`
- `flex-end`
- `space-around`
- `space-between`

Try changing `justify-content` to `flex-start` and `flex-end`. This should align the menu to the left and right side of the browser window, respectively. Be sure to change it back to `center` before moving on. The last two options are only useful when you have multiple flex items in a container.

- `center`
- `flex-start`
- `flex-end`
- `space-around`
- `space-between`

Try changing `justify-content` to `flex-start` and `flex-end`. This should align the menu to the left and right side of the browser window, respectively. Be sure to change it back to `center` before moving on. The last two options are only useful when you have multiple flex items in a

container.

DISTRIBUTING MULTIPLE FLEX ITEMS

“Big deal,” you might be saying: we can do left/right alignment with floats

- `space-between`

Try changing `justify-content` to `flex-start` and `flex-end`. This should align the menu to the left and right side of the browser window, respectively. Be sure to change it back to `center` before moving on. The last two options are only useful when you have multiple flex items in a container.

DISTRIBUTING MULTIPLE FLEX ITEMS

“Big deal,” you might be saying: we can do left/right alignment with floats and centering with auto-margins. True. Flexbox doesn’t show its real strength until we have more than one item in a container. The `justify-content` property also lets you distribute items equally inside a container.

SPACE-AROUND SPACE-BETWEEN

DISTRIBUTING MULTIPLE FLEX ITEMS

“Big deal,” you might be saying: we can do left/right alignment with floats and centering with auto-margins. True. Flexbox doesn’t show its real strength until we have more than one item in a container. The `justify-content` property also lets you distribute items equally inside a container.

SPACE-AROUND SPACE-BETWEEN

Change our `.menu` rule to match the following:

DISTRIBUTING MULTIPLE FLEX ITEMS

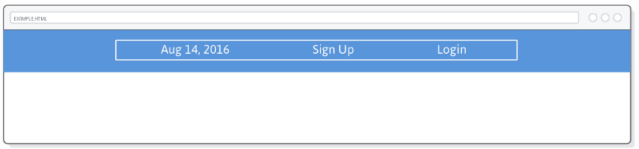
“Big deal,” you might be saying: we can do left/right alignment with floats and centering with auto-margins. True. Flexbox doesn’t show its real strength until we have more than one item in a container. The `justify-content` property also lets you distribute items equally inside a container.

SPACE-AROUND SPACE-BETWEEN

Change our `.menu` rule to match the following:

```
.menu {
  border: 1px solid #fff;
  width: 900px;
  display: flex;
  justify-content: space-around;
}
```

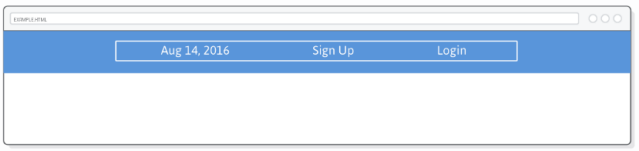
This turns our `.menu` into a nested flex container, and the `space-around` value spreads its items out across its entire width. You should see something like this:



The flex container automatically distributes extra horizontal space to either side of each item. The `space-between` value is similar, but it only adds that extra space *between* items. This is what we actually want for our example page, so go ahead and update the `justify-content` line:

```
justify-content: space-between;
```

Of course, you can also use `center`, `flex-start`, `flex-end` here if you want to push all the items to one side or another, but let's leave it as `space-between`.



The flex container automatically distributes extra horizontal space to either side of each item. The `space-between` value is similar, but it only adds that extra space *between* items. This is what we actually want for our example page, so go ahead and update the `justify-content` line:

```
justify-content: space-between;
```

Of course, you can also use `center`, `flex-start`, `flex-end` here if you want to push all the items to one side or another, but let's leave it as `space-between`.

GROUPING FLEX ITEMS

side of each item. The `space-between` value is similar, but it only adds that extra space *between* items. This is what we actually want for our example page, so go ahead and update the `justify-content` line:

```
justify-content: space-between;
```

Of course, you can also use `center`, `flex-start`, `flex-end` here if you want to push all the items to one side or another, but let's leave it as `space-`

between.

GROUPING FLEX ITEMS

Flex containers only know how to position elements that are one level deep (i.e., their child elements). They don't care one bit about what's inside their flex items. This means that grouping flex items is another weapon in your layout-creation arsenal. Wrapping a bunch of items in an extra `<div>` results in a totally different web page.

to push all the items to one side or another, but let's leave it as `space-between`.

GROUPING FLEX ITEMS

Flex containers only know how to position elements that are one level deep (i.e., their child elements). They don't care one bit about what's inside their flex items. This means that grouping flex items is another weapon in your layout-creation arsenal. Wrapping a bunch of items in an extra `<div>` results in a totally different web page.

NO GROUPING
(3 FLEX ITEMS)

GROUPED ITEMS
(2 FLEX ITEMS)

For example, let's say you want both the **Sign Up** and **Login** links to be on the right side of the page, as in the screenshot below. All we need to do is stick them in another `<div>`:

Flex containers only know how to position elements that are one level deep (i.e., their child elements). They don't care one bit about what's inside their flex items. This means that grouping flex items is another weapon in your layout-creation arsenal. Wrapping a bunch of items in an extra `<div>` results in a totally different web page.

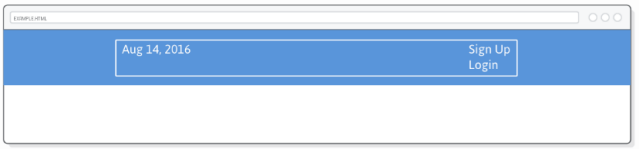
NO GROUPING
(3 FLEX ITEMS)

GROUPED ITEMS
(2 FLEX ITEMS)

For example, let's say you want both the **Sign Up** and **Login** links to be on the right side of the page, as in the screenshot below. All we need to do is stick them in another `<div>`:

```
<div class='menu'>
  <div class='date'>Aug 14, 2016</div>
  <div class='links'>
    <div class='signup'>Sign Up</div>      <!-- This is nested now -->
    <div class='login'>Login</div>         <!-- This one too! -->
  </div>
</div>
```

Instead of having three items, our `.menu` flex container now has only two (`.date` and `.links`). Under the existing `space-between` behavior, they'll snap to the left and right side of the page.

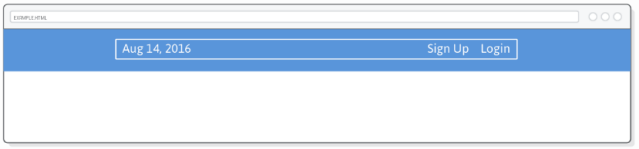


But, now we need to lay out the `.links` element because it's using the default block layout mode. The solution: more nested flex containers! Add a new rule to our `styles.css` file that turns the `.links` element into a flex container:

```
.links {
  border: 1px solid #fff;  /* For debugging */
  display: flex;
  justify-content: flex-end;
}

.login {
  margin-left: 20px;
}
```

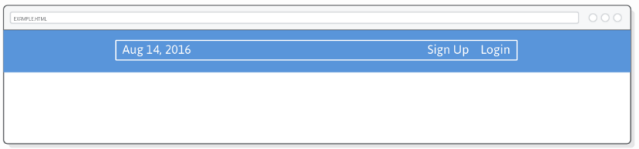
This will put our links right where we want them. Notice that margins still work just like they did in the CSS Box Model. And, as with the normal box model, auto margins have a special meaning in flexbox (we'll leave that for the end of the chapter though).



We won't need those white borders anymore, so you can go ahead and delete them if you like.

This will put our links right where we want them. Notice that margins still work just like they did in the CSS Box Model. And, as with the normal box model, auto margins have a special meaning in flexbox (we'll leave that for the end of the chapter though).

We won't need those white borders anymore, so you can go ahead and delete them if you like.



We won't need those white borders anymore, so you can go ahead and delete them if you like.

CROSS-AXIS (VERTICAL) ALIGNMENT

So far, we've been manipulating horizontal alignment, but flex containers can also define the vertical alignment of their items. This is something that's simply not possible with floats.

CROSS-AXIS (VERTICAL) ALIGNMENT

So far, we've been manipulating horizontal alignment, but flex containers can also define the vertical alignment of their items. This is something that's simply not possible with floats.

JUSTIFY-CONTENT

ALIGN-ITEMS

To explore this, we need to add a header underneath our menu. Add the following markup to `flexbox.html` after the `.menu-container` element:

```
<div class='header-container'>
  <div class='header'>
    <div class='subscribe'>Subscribe &#9662;</div>
    <div class='logo'><img src='images/awesome-logo.svg'/></div>
    <div class='social'><img src='images/social-icons.svg'/></div>
  </div>
</div>
```

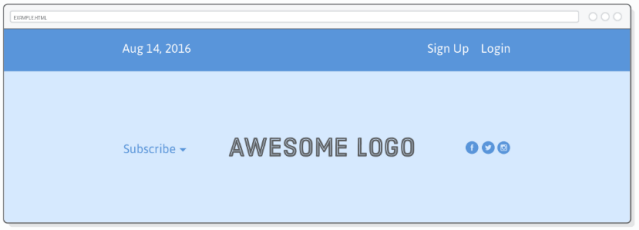
Next, add some base styles to get it aligned with our `.menu` element:

```
.header-container {
  color: #5995DA;
  background-color: #D6E9FE;
  display: flex;
  justify-content: center;
}

.header {
  width: 900px;
  height: 300px;
  display: flex;
  justify-content: space-between;
}
```

This should all be familiar; however, the scenario is a little bit different than our menu. Since `.header` has an explicit height, items can be positioned

vertically inside of it. The official specification calls this “cross-axis” alignment (we’ll see why in a moment), but for our purposes it might as well be called “vertical” alignment.



Vertical alignment is defined by adding an `align-items` property to a flex container. Make our example page match the above screenshot with the following line:

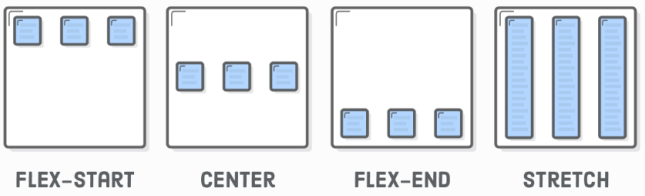
our menu. Since `.header` has an explicit height, items can be positioned vertically inside of it. The official specification calls this “cross-axis” alignment (we’ll see why in a moment), but for our purposes it might as well be called “vertical” alignment.

Vertical alignment is defined by adding an `align-items` property to a flex container. Make our example page match the above screenshot with the following line:

```
.header {
  /* ... */
  align-items: center;  /* Add this */
}
```

The available options for `align-items` is similar to `justify-content`:

- `center`
- `flex-start` (top)
- `flex-end` (bottom)
- `stretch`
- `baseline`



Most of these are pretty straightforward. The `stretch` option is worth a taking a minute to play with because it lets you display the background of each element. Let’s take a brief look by adding the following to `styles.css`:

- `center`
- `flex-start` (top)
- `flex-end` (bottom)
- `stretch`
- `baseline`

Most of these are pretty straightforward. The `stretch` option is worth a taking a minute to play with because it lets you display the background of each element. Let's take a brief look by adding the following to `styles.css`:

```
.header {
  /* ... */
  align-items: stretch;    /* Change this */
}

.social,
.logo,
.subscribe {
  border: 1px solid #5995DA;
}
```

The box for each item extends the full height of the flex container, regardless of how much content it contains. A common use case for this behavior is creating equal-height columns with a variable amount of content in each one—something very difficult to do with floats.

Be sure to delete the above changes and vertically center our content inside of `.header` before moving on.

WRAPPING FLEX ITEMS

Flexbox is a more powerful alternative to float-based grids. Not only can it

The box for each item extends the full height of the flex container, regardless of how much content it contains. A common use case for this behavior is creating equal-height columns with a variable amount of content in each one—something very difficult to do with floats.

Be sure to delete the above changes and vertically center our content inside of `.header` before moving on.

WRAPPING FLEX ITEMS

Flexbox is a more powerful alternative to float-based grids. Not only can it render items as a grid—it can change their alignment, direction, order, and size, too. To create a grid, we need the `flex-wrap` property.

The box for each item extends the full height of the flex container, regardless of how much content it contains. A common use case for this behavior is creating equal-height columns with a variable amount of content in each one—something very difficult to do with floats.

Be sure to delete the above changes and vertically center our content inside of `.header` before moving on.

WRAPPING FLEX ITEMS

Flexbox is a more powerful alternative to float-based grids. Not only can it render items as a grid—it can change their alignment, direction, order, and size, too. To create a grid, we need the `flex-wrap` property.

The box for each item extends the full height of the flex container, regardless of how much content it contains. A common use case for this behavior is creating equal-height columns with a variable amount of content in each one—something very difficult to do with floats.

Be sure to delete the above changes and vertically center our content inside of `.header` before moving on.

WRAPPING FLEX ITEMS

Flexbox is a more powerful alternative to float-based grids. Not only can it render items as a grid—it can change their alignment, direction, order, and size, too. To create a grid, we need the `flex-wrap` property.

WRAPPING FLEX ITEMS

Flexbox is a more powerful alternative to float-based grids. Not only can it render items as a grid—it can change their alignment, direction, order, and size, too. To create a grid, we need the `flex-wrap` property.

NO WRAPPING

FLEX-WRAP: NOWRAP;

WITH WRAPPING

FLEX-WRAP: WRAP;

Add a row of photos to `flexbox.html` so that we have something to work with. This should go inside of `<body>`, under the `.header-container` element:

```
<div class='photo-grid-container'>
  <div class='photo-grid'>
    <div class='photo-grid-item first-item'>
      <img src='images/one.svg'/>
    </div>
    <div class='photo-grid-item'>
      <img src='images/two.svg'/>
    </div>
    <div class='photo-grid-item'>
      <img src='images/three.svg'/>
    </div>
  </div>
</div>
```

Again, the corresponding CSS should be familiar from previous sections:

```
.photo-grid-container {
  display: flex;
```

```
  justify-content: center;
}

.photo-grid {
  width: 900px;
  display: flex;
  height: 300px;
}
```

This should work as expected, but watch what happens when we add more items than can fit into the flex container. Insert an extra two photos into the `.photo-grid`:

```
<div class='photo-grid-item'>
  <img src='images/four.svg'/>
</div>
<div class='photo-grid-item last-item'>
  <img src='images/five.svg'/>
</div>
```

By default, they flow off the edge of the page: