A More Complex Layout

We're building a layout that resembles what you may build in a real situation. Here's where the real fun begins.

There is a lot to cover in this chapter. We will split it into two parts to thoroughly explain how you could do the similar things in different ways with Susy.

In this chapter, you will learn:

- When to use Susy, and when not to use Susy
- How to center a Susy grid item with the `span()` mixin
- How to center a Susy grid item with the `span()` and `gutter()` functions

In the next chapter, you will learn:

- How to create a gallery with the `span()` mixin
- How to create a gallery with the `gallery()` mixin

Here's what we're building in these two chapters:

Taking Care of CSS

We're speeding up the CSS again this chapter. There are some nuggets here that you might want to add into your own styles.

Here's what I did in addition to giving height and background colors to the layouts:

- removed margins and paddings from all `<ul>` , `<ol>` and `<li>` elements
- removed list styles from all `<li>` elements
- added `max-width: 100%` and `height: auto` to make images responsive.

```
// Scss
h2 {
  padding: 1rem 0;
  text-align: center;
  color: #555;
}

ul, ol {
  margin: 0;
  padding: 0;
}

li {
    list-style: none;
}
img {
  max-width: 100%;
  height: auto;
}

.site-header,.site-footer {
  background: rgba(234, 159, 195, 0.8);
}
```

```
.site-header {
  a {
    color: #555;
    text-decoration: none;
  }
}

.content {
  margin-top: 5vh;
  padding-bottom: 1rem;
  background: rgba(113, 218, 210, 0.8);
}

.sidebar {
  margin-top: 5vh;
  background: rgba(250, 231, 179, 0.8);
  padding-bottom: 1rem;
  a {
    color: #666;
    padding-left: 1rem;
    line-height: 2;
    text-decoration: none;
  }
}

.widget {
  background: rgba(240, 150, 113, 0.8);
}

.site-footer {
  margin-top: 5vh;
}
```

The Susy Map

As with all Susy projects, you begin with the `$susy` map.

You can see from the layout that there are a total of 16 columns in this project. Everything else remains the same as with the previous layout.

```
// Scss
$susy: (
  columns: 16,
  container: 1140px,
  global-box-sizing: border-box,
  debug: (image: show)
  );

@include border-box-sizing;

.wrap {
  @include container();
}
```

The Header Section

Let's break down the HTML as we go along, beginning with the header.

The header contains a logo and some navigational links. It has a background, which takes up 100% of the browser width. In order for this to happen, we have to keep the grid container (`.wrap`) within the `<header>` .

```
<!-- HTML -->
<header class="site-header">
  <div class="wrap"></div>
</header>
```

We can see that the logo is flushed to the left of the grid while the navigational links are flushed towards the right of the grid. This would mean that both the `.logo` and `<nav>` are wrapped within `.wrap` .

```
<!-- HTML -->
<header class="site-header">
  <div class="wrap">
    <div class="logo"><a href="#">Logo</a></div>
    <nav>
      <ul>
        <li><a href="#">item01</a></li>
        <li><a href="#">item02</a></li>
        <li><a href="#">item03</a></li>
        <li><a href="#">item04</a></li>
        <li><a href="#">item05</a></li>
      </ul>
    </nav>
  </div>
</header>
```

Since the logo is flushed left and the navigational links are flushed right, there is no need to use Susy to create their styles. We can simply apply `float: left` to `.logo` and a `float: right` to `<nav>`.

```
.logo {
  float: left;
  line-height: 2rem;
  font-size: 1.5rem;
}

nav {
  float: right;

  li {
    list-style: none;
    float: left;
    margin-left: 1em;
    line-height: 2rem;
  }
}
```

View Source Code

As you can see, you don't have to use Susy with every element. You can always use standard CSS if Susy becomes overkill.

The Content and Sidebar

The `.content` and `.sidebar` sections are similar to what we had in the previous chapter. This time round, let's add some html elements into these sections.

```
<!-- HTML -->
  <div class="wrap">
    <main class="content">
      <h2>Gallery Content</h2>
      <ul class="gallery">
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
```

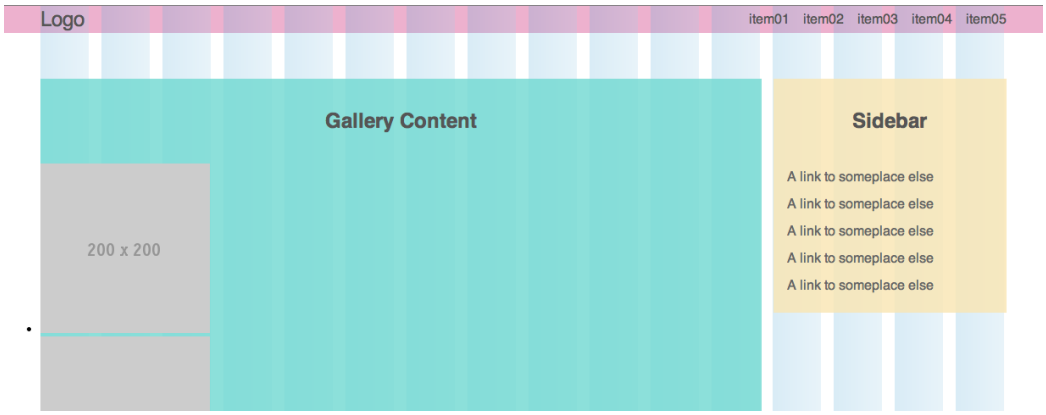
```
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
        <li class="gallery__item"><img
src="http://www.placehold.it/300x300" alt="">
        </li>
      </ul>
    </main>
    <aside class="sidebar">
      <h2>Sidebar</h2>
      <ul>
        <li><a href="#">A link to someplace else</a>
        </li>
        <li><a href="#">A link to someplace else</a>
        </li>
        <li><a href="#">A link to someplace else</a>
        </li>
        <li><a href="#">A link to someplace else</a>
        </li>
        <li><a href="#">A link to someplace else</a>
        </li>
      </ul>
    </aside>
```

```
  </div>
```

Sass code for `.content` and `.sidebar` would be as follows since `.content` takes up 12 of 16 columns and `.sidebar` takes up 4 of 16 columns:

```
// Scss
.content {
  @include span(12 of 16);
}

.sidebar {
  @include span (4 of 16 last);
}
```



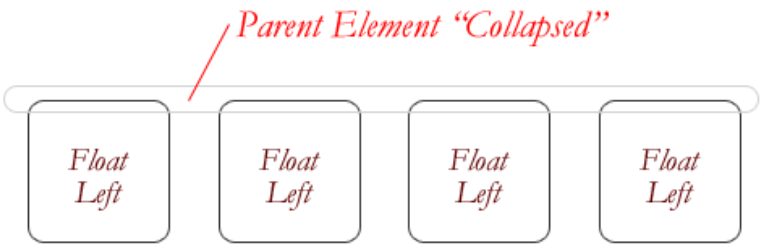
View Source Code

The Gallery Within Content

Susy layouts are predominantly made with floats. This brings me to the float collapse problem. It happens when every child item within the parent element is floated. Chris Coyier explains this problem the best:

> “One of the more bewildering things about working with floats is how they can affect the element that contains them (their “parent”

element). If this parent element contained nothing but floated elements, the height of it would literally collapse to nothing. This isn't always obvious if the parent doesn't contain any visually noticeable background, but it is important to be aware of".



"Collapsing almost always needs to be dealt with to prevent strange layout and cross-browser problems. We fix this problem by clearing the float after the floated elements in the container but before the close of the container".

All About Floats, Chris Coyier

The parent element's height will collapse to nothing all of its child elements are floated. We have to fix this float collapse problem with a `clearfix` mixin.

This `clearfix` mixin will include a pseudo element that will clear the floated contents within the container right before the container closes.

Place this mixin at the beginning of your Sass file:

```
// Scss. Clearfix Mixin
@mixin cf {
  &:after {
    content: " ";
    display: block;
    clear: both;
  }
}
```

We know that `.gallery` is the container for `.gallery__items` and that we are going to use the `span()` mixin on each `.gallery__item`. This means that all child elements within `.gallery` are floated. We will need to give `.gallery` a clearfix.

```
// Scss
.gallery {
  @include cf;
}
```

Before working on the rest of `.gallery`, let's take a look at the final layout again:

From the image, you can see that the `gallery__items` fit onto a 10 column width. This would mean that we need to position `.gallery` in such a way that it is centered within `.content`, and that its width takes up 10 columns exactly.

There are two ways to center `.gallery` in the middle of the 12-column `.content`. We will walk through all both ways. Before that do that, we need to learn more about other Susy functions that Susy provides us with.

These functions can help us tremendously in achieving what we want. They are the `span()` function and the `gutter()` function.

The `span()` Function

You may be thinking, isn't `span` a mixin?

That is correct. But Susy also provides a function that uses the same name. The `span` function takes in exactly the same arguments as the `span` mixin, but it returns only the value of `width` instead of writing 3 properties.

Here's a comparison between the `span` function and the `span` mixin in use:

```
// SCSS
.span-mixin {
  @include span(12 of 16);
}

.span-function {
  width: span(12 of 16);
}
```

```
/* CSS */
.span-mixin {
  width: 74.68354%;
  float: left;
  margin-right: 1.26582%;
}

.span-function {
  width: 74.68354%;
}
```

Note: If you're unsure of how to work with functions and mixins, turn back to chapter 3, it's explained there :)

The span function returns the width of the desired `$span` and can be used in any property. Here's an example where the function is used on a `margin` property:

```
// SCSS
.span-function {
  margin-right: span(1);
}
```

```
/* CSS */
.span-function {
  margin-right: 5.06329%;
}
```

Here, we've created a `margin-right` property and given it a value equal to one column.

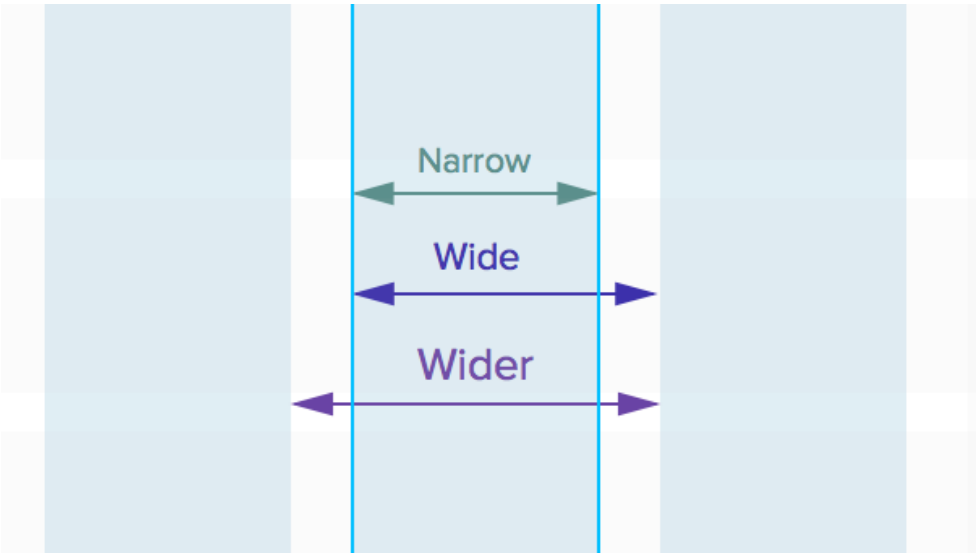
There is one more useful argument that you can give to both the `span()` function and the `span()` mixin: `$spread`.

The `$spread` Argument

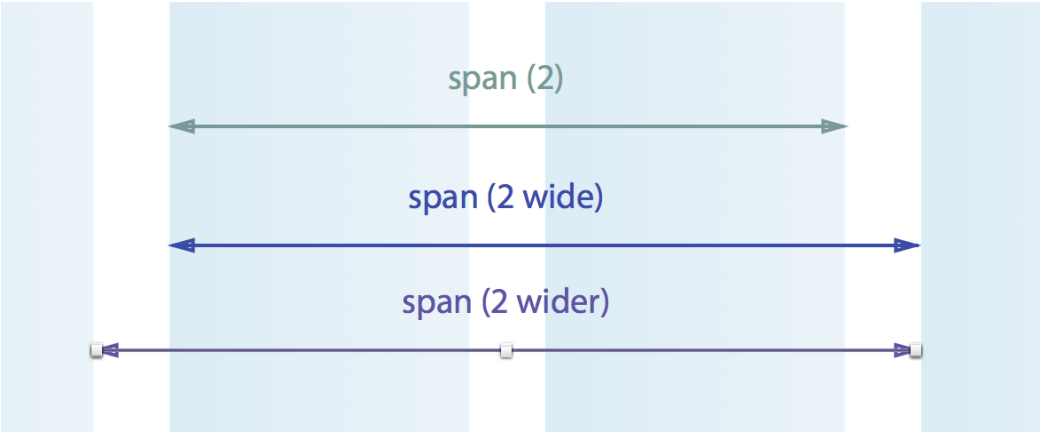
`$spread` is an optional argument that allows you to change the `width` output in both the `span()` function and mixin to explicitly state whether the `width` should be expanded to include one or two more gutters.

`$spread` has 3 different options for you to choose from.

- Narrow (*default*)
- Wide
- Wider



When calculating the width of a `span`, Susy will include all internal gutters by default. `wide` and `wider` simply adds one or two more gutters into the width.



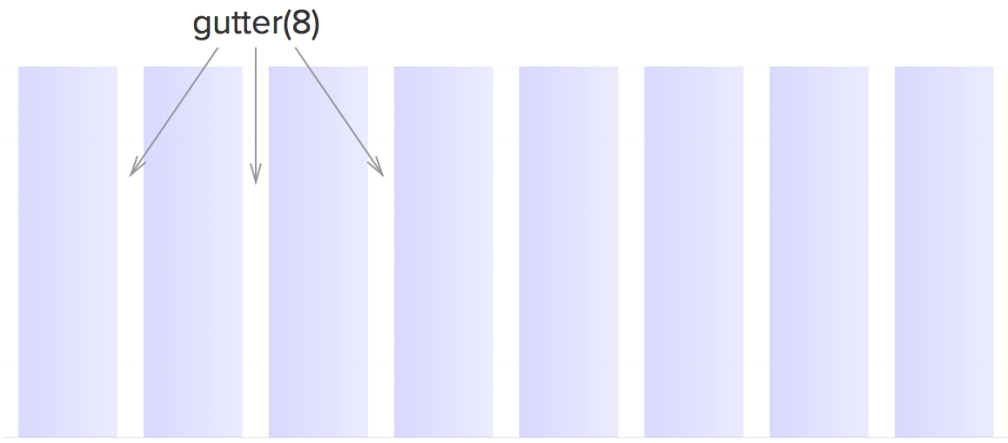
The `narrow` option works best most of the time. You'll only use `wide` or `wider` if you need to add a gutter or two to the `width` calculations.

The `gutter()` Function

The `gutter()` function, as its name suggests, is a function to output the width of a gutter. It takes one argument: `$context`.

```
// Scss
.test {
  width: gutter($context);
}
```

This `$context` allows the `gutter()` function to calculate and output the width of one gutter. In a container with 8 columns, one gutter size would be `gutter(8)`.



If you didn't give the `gutter` function a context, it will look for the context in other areas as explained in the previous chapter.

LEt's see how we can use these two functions to center the layout now.

Centering the Gallery

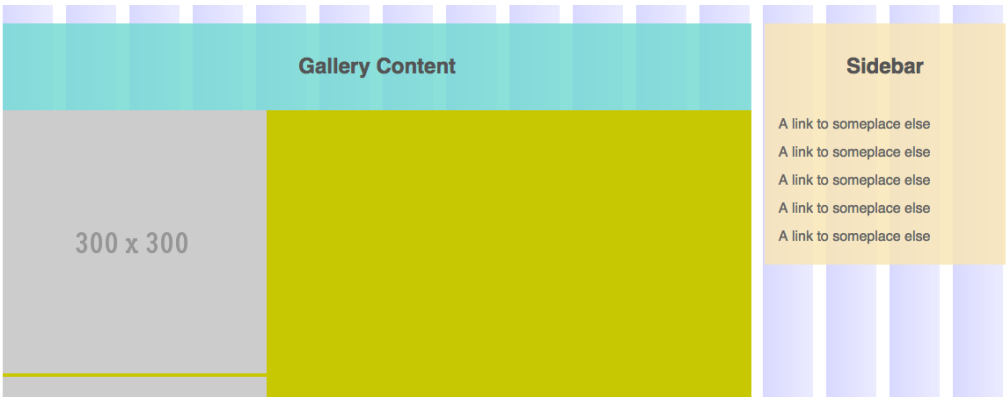
There are two ways to center `.gallery`.

1. Setting the width of `.gallery`
2. Adding padding to `.gallery`

Both methods work perfectly fine. Feel free to use the two of them interchangeably.

Since we're working on centering the gallery for the first time, let's give `.gallery` a temporary background color of `rgb(200,200,2)` to detect its exact size and position.

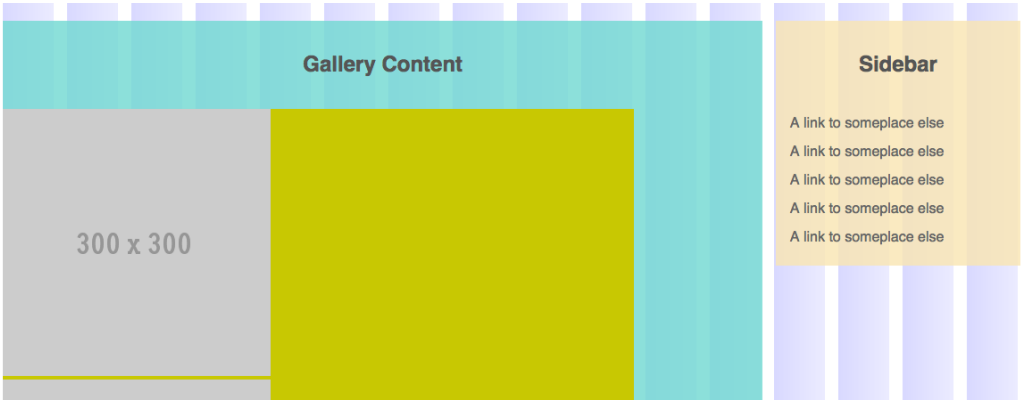
```
// Scss
.gallery {
  @include cf;
  background: rgb(200, 200, 2);
}
```



Method 1: Setting the Width of Gallery

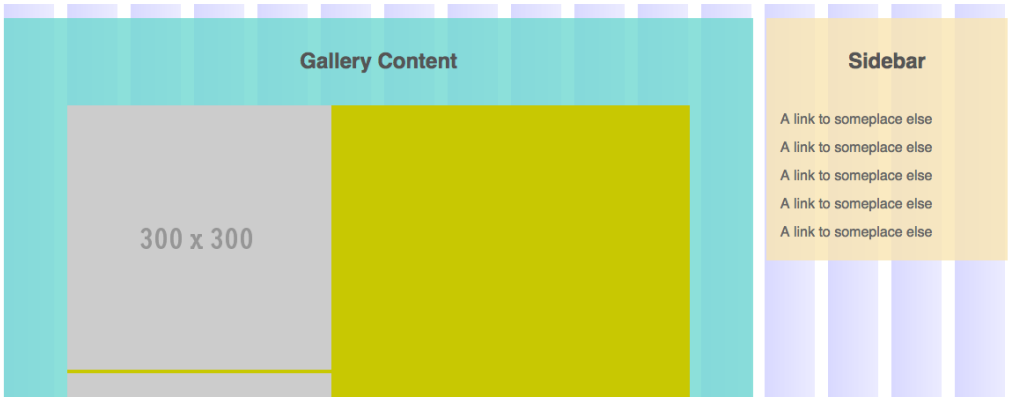
The first method is to set the width of the parent container to 10 columns. We can do this with either the `span()` mixin or span function. Let's use the `span()` function to keep the code DRY since we only require the `width` property.

```
// Scss
.gallery {
  width: span(10 of 12);
}
```



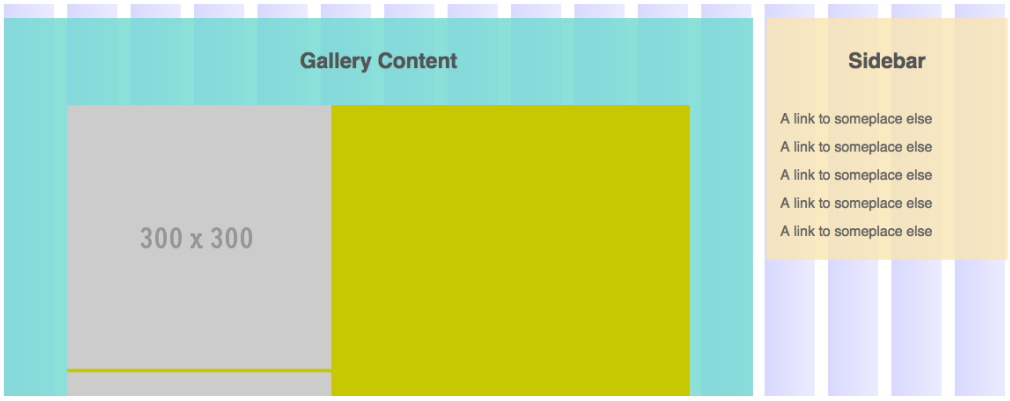
We can center the container the same way as we centered the `.wrap` container by setting `margin-left` and `margin-right` to auto.

```
// Scss
.gallery {
  width: span(10 of 12);
  margin-left: auto;
  margin-right: auto;
}
```



Alternatively, since we know the context, we can push the `.gallery` from one side of the container with the `span` function, or with a combination of the `span()` and `gutter()` functions.

```
// Scss
.gallery {
  width: span(10 of 12);
  margin-left: span(1 wide of 12);
  // OR
  // margin-left: span(1 of 12) + gutter(12);
}
```



View Source Code

Method 2: Adding Padding to Gallery

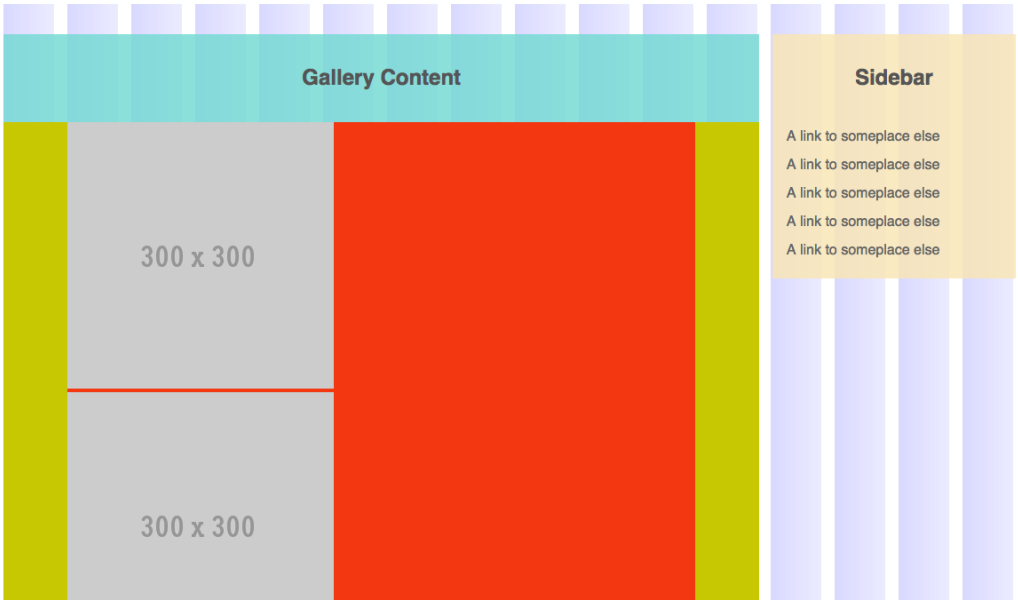
The second method is to add `padding-left` and `padding-right`, that is the width of one column plus one gutter, to `.gallery`. This method only works if you are using the `border-box` box-sizing property, which we are if you've been following along.

If we choose to add padding to the gallery, we have to add a background to `.gallery__item` instead to visualize how much space `.gallery__item` has at 100% width.

```
// Scss
.gallery__item {
  background: rgb(100, 100, 200);
}
```

Again, we can either use a combination of the `span()` and `gutter()` functions or we can use the `span()` function with the `$spread` keyword.

```
// Scss
.gallery {
  padding: 0 span(1 wide of 12);
  // OR
  // padding: 0 span(1 of 12) + gutter(12);
}
```



View Source code

A Quick Wrap Up

We have gone through the basics of creating a more complex layout in this chapter. We have also learned how to center any HTML element within Susy. You may also have discovered that there are multiple ways of achieving the same outcome when using Susy, and that makes Susy flexible enough to adapt to your unique requirements.

We will move on to complete the layout in the next chapter and you will learn more about how to create galleries with Susy.