3 DE SEPTIEMBRE DE 2019 / #DISEÑO DE RESPUESTA

Cómo pensar de manera receptiva: un tutorial de diseño web receptivo

kevin powell

Durante mucho tiempo, el diseño web receptivo fue una tendencia. Ahora es simplemente una realidad. Si pensamos en un sitio web, realmente no tenemos que decir "un sitio web receptivo", es solo una realidad esperada.

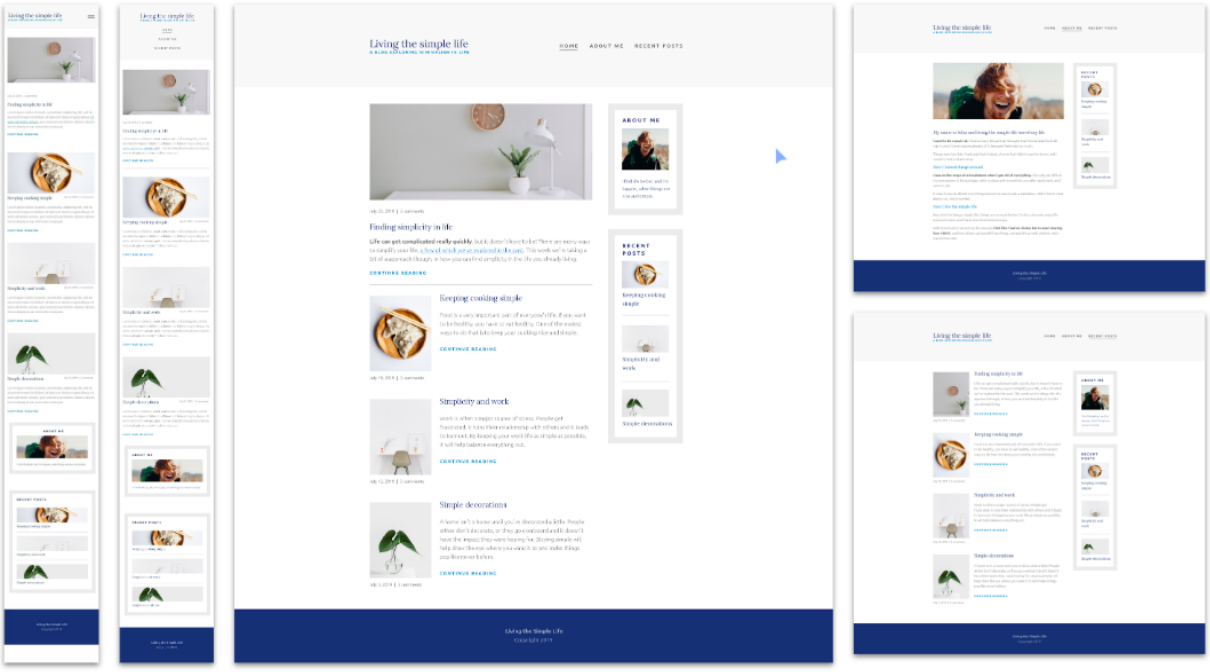
Esto significa que cuando estamos armando un sitio web, debe construirse teniendo en cuenta cómo se verá en diferentes tamaños de pantalla. Con la tendencia actual, debe haber un fuerte énfasis en la experiencia móvil.

R ecientemente , lancé un curso completo y detallado sobre ***Scrimba*** *llamado* ***The Responsive Web Design Bootcamp*** **.** *Este curso cubre todos los conceptos técnicos y arquitectónicos sobre el diseño web receptivo en gran profundidad* **.** *Este blog se basa en una de las seis secciones del curso :* ***Comenzar a pensar de manera receptiva*** *.*

Entrar en el estado de ánimo receptivo

1. Cómo abordar un diseño
2. Unidades CSS
3. Conceptos básicos de la caja flexible
4. Conceptos básicos de consulta de medios

Si bien usamos algunos ejercicios simples para comenzar, el enfoque principal de este módulo del curso es construir un sitio web de 3 páginas que responda completamente. Lo usamos tanto para reforzar lo que ya hemos aprendido como para introducir algunas cosas nuevas en la mezcla.



En esta publicación, exploraremos los conceptos que exploro en este módulo del curso que se enumeran anteriormente, desde una inmersión en `em` vs `rem`, una mirada a los conceptos básicos de flexbox y media queries, así como una descripción general de algunos de los proyectos que construimos en el curso.

El primer y más fundamental concepto en la construcción de un diseño web receptivo son las unidades que usamos para configurar muchas de nuestras propiedades.

En la lección 2 vamos a aprender cuáles son algunas de las unidades de CSS disponibles y en qué se diferencian entre sí. Lo más importante es que aprenderemos cuáles usar según los requisitos.

Hay tres tipos principales de unidades CSS disponibles: Unidades absolutas, Unidades de porcentaje y Unidades relativas.

Unidades absolutas

Las unidades absolutas también se denominan unidades fijas.

La longitud expresada en cualquiera de las unidades absolutas aparecerá exactamente en el mismo tamaño (por eso las llamamos fijas, son de tamaño fijo).

Si bien `px` (píxeles) son la unidad más común, en CSS también podemos usar `pt`, `pc`, `in`, `cm`, `mm` y muchos más, aunque realmente no los recomendaría a menos que esté diseñando algo para imprimir.

`px` son un poco más complicados de lo que piensas. En los viejos tiempos, a `px` estaba relacionado con un píxel en su pantalla, pero ahora CSS usa algo llamado *píxel de referencia* que lo convierte en un tamaño fijo, independiente de la resolución del dispositivo.

Porcentajes

Las lecciones 3-5 del curso se sumergen en el porcentaje. El porcentaje, como sugiere su nombre, se usa a menudo para definir el tamaño en relación con el tamaño de su padre.

```
.box {
    width: 500px;
    /* this element will be 500px in width */
}
```

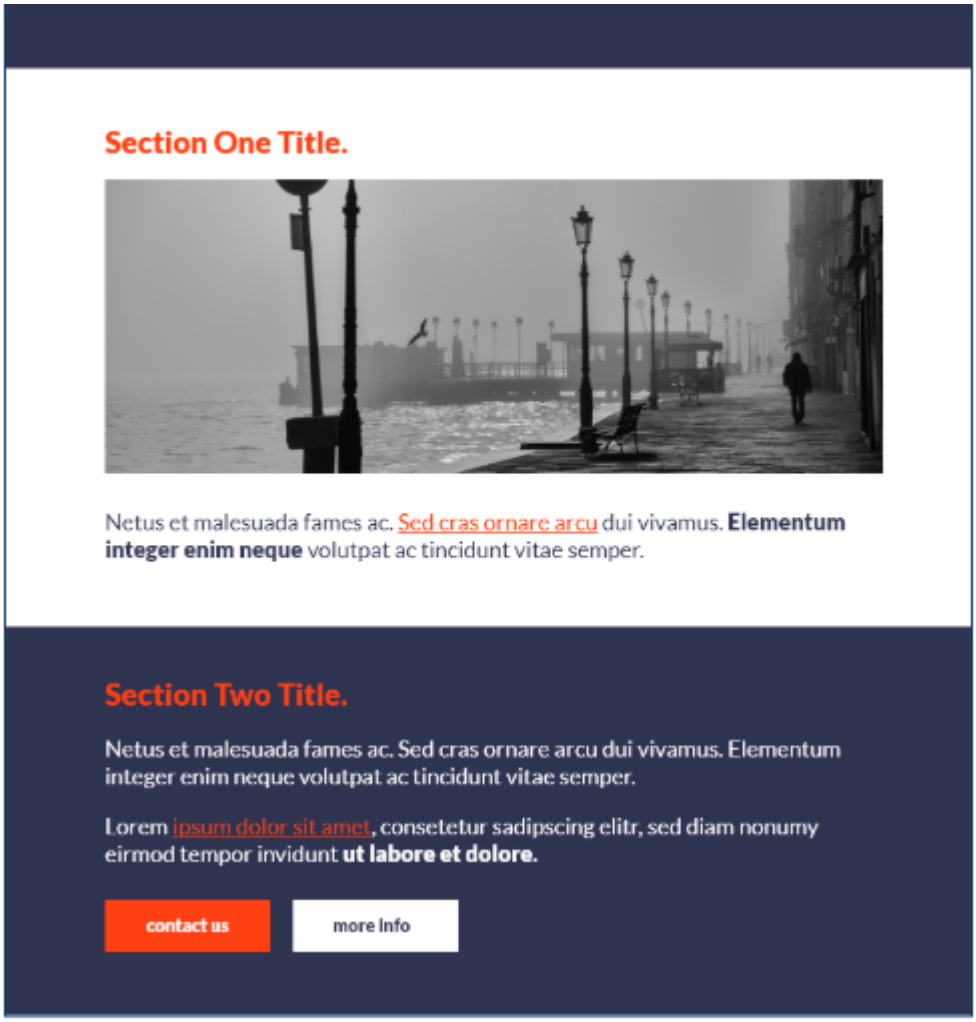
Mientras que, si usamos un porcentaje, no es tan sencillo:

```
.box {
    width: 80%;
    /* this element will have a width of 80% */
}
```

Pero, ¿de qué es eso `80%` ? Es de su padre. Cuando usamos porcentaje para `width` , `margin` o `padding` siempre está mirando el *ancho de su padre* (sí, incluso para `margin` y `padding` en la parte superior e inferior).

A menudo usamos el porcentaje para definir el ancho de los elementos, ya que les permite ser más flexibles, lo cual es esencial al armar un diseño receptivo.

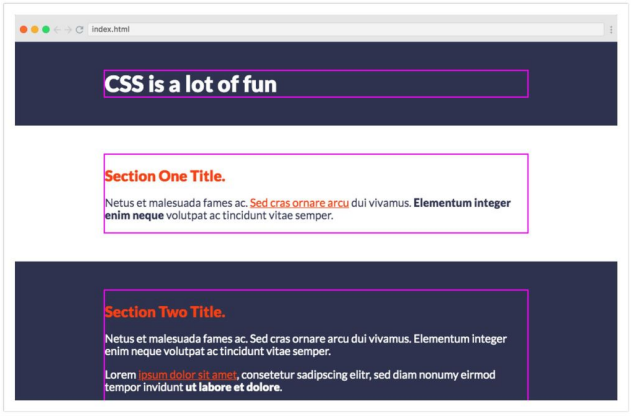
Durante el curso, aprenderemos esto con la ayuda del siguiente ejemplo:



Si mantenemos el ancho del contenedor para el diseño anterior en un número fijo, digamos `500px`, sabemos que se verá bien a menos que el tamaño de la pantalla sea menor que `500px`.

En este escenario, si queremos que nuestro diseño se ajuste según el tamaño de la ventana gráfica, lo que podemos hacer es establecer el ancho en porcentaje, que puede ser `100%` o menos.

```
.container {
    width: 70%;
    ...
}
```

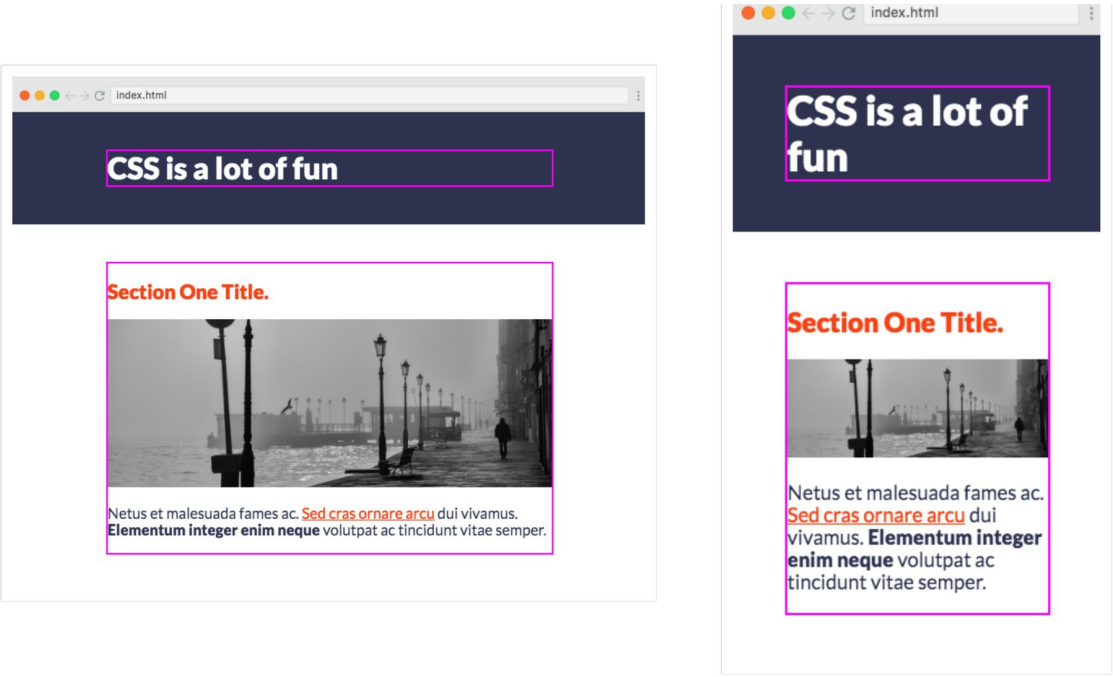


Sin embargo hay un problema. En este momento, si ponemos una imagen dentro de nuestro contenedor, se ve así (si la imagen tiene un ancho demasiado grande):

El problema es que, a menos que especifiquemos lo contrario, las imágenes tendrán su tamaño real. Para solucionar esto, podríamos aplicar CSS en la imagen misma para que comience a tener el tamaño predeterminado de su contenedor.

```
.img {
    width: 100%;
}
```

Y, mientras hacemos esto, podemos ver que la imagen comienza a tener el tamaño predeterminado de su contenedor principal.



Tamaños máximos y mínimos

Si bien la solución anterior es excelente, en realidad puede hacer que la imagen se vuelva más grande que su tamaño real. Si esto sucede, rápidamente comenzará a perder calidad.

Para ayudar a solucionar eso, podemos usar `max-width`.

```
.img {
    max-width: 100%;
}
```

Esto significa que el ancho máximo de la imagen es el 100% de su padre, pero se permite que sea más pequeño.

En el curso, también exploro otras propiedades similares, como `min-width`, y `min-height` y `max-height`, que establecen límites superior e inferior en el tamaño de un elemento.

Unidades CSS - Unidades relativas

Relative units in CSS are always relative on the size of something else. Some units will look at the `font-size` of another element (or that element), while others will look at the size of the view port.

In the course, I start by introducting the two which are relative to `font-size` , the **em** and **rem**.

The `em` unit

When you declare the `font-size` of an element, if you use `em` s, it will be relative to the parent's `font-size` .

For example, we have the following HTML code where we have some `li` elements which have few parent elements.

```
<body>
    <section class='class-one'>
        <div class="container">
            <ul>
                <li>Item 1</li>
                <li>Item 2</li>
                <li>Item 3</li>
            </ul>
        </div>
    </section>
</body>
```

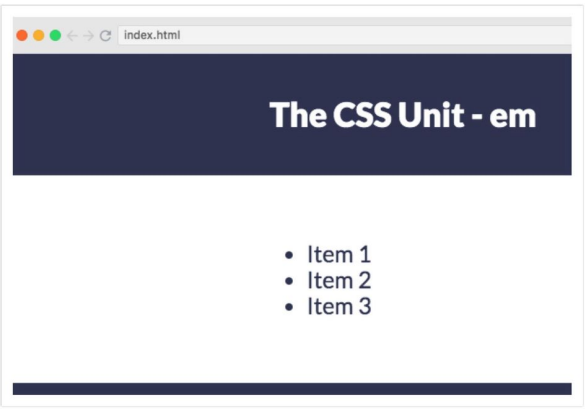
And, we have the following CSS with the above HTML:

```
body {
    font-size: 25px; ...
}
```

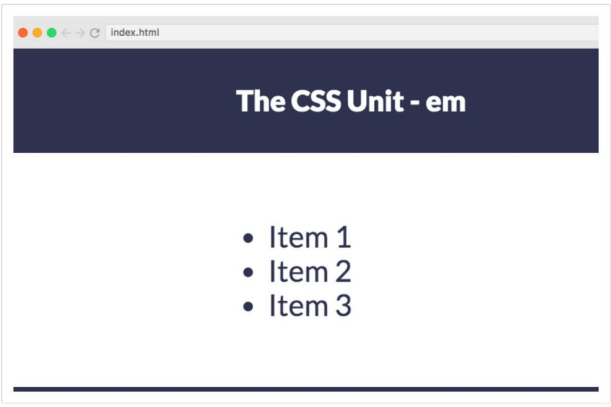
```
}
```

In the above case, the `ul` will take its `font-size` from its parent and apply on its `li` elements using the relative `1.5em`.

ul: 1em



ul: 1.5em



In case of *em* unit, we have to remember that it will default to the `font-size` of its immediate parent, which means that if in our above example, we change the `font-size` of class `container` to be `15px`, then the `ul` will start to size relative `15px`.

Cascading effect of `em`s

One of the reasons I don't like to set `font-size` in `em` is because of the risk of it being hit by a cascading effect. For example, if we use the above example, but add in a `font-size` to the `li` elements, we start to run into problems!

```
body {
    font-size: 25px; ...
}

ul {
    font-size: 1.5em;
```

```
}
```

In this case, the `li` looks at it's parent, which then looks at the body, so we get a total font-size of 56.25px (1.5 x 1.5 x 25px). Things can quickly get out of control!

The rem unit

As we can see, using `em` can be confusing at times since it takes it's size from its parent which may have its `font-size` defined also in `em` and that would create a cascading effect.

The `rem` unit is short for *Root Em*, meaning that it only looks at the `font-size` of the root element. In the case of websites, the root element is always the `html` element.

So if you set the font-size of something in `rem` it is *always* relative to the `font-size` of the `html` element and nothing else, making it much easier to use.

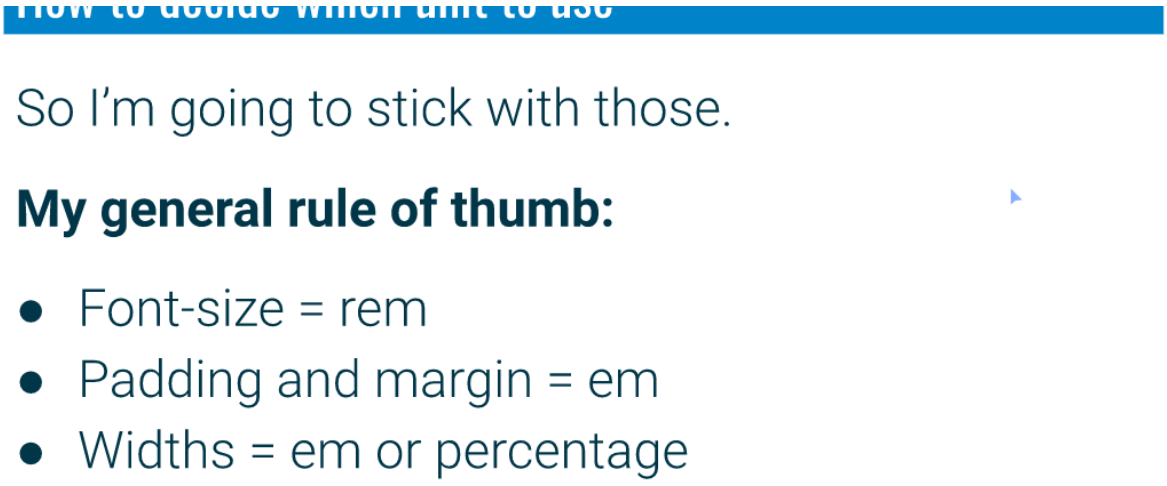
Is it ever a good idea to use `em`s?

While `em` might seem like something you might want to avoid, they can be really useful!

When setting `font-size` with `em`, it looks to the parent element, but when we set any other property using `em`, it is relative to **that element's font-size!** That's super useful for setting things like `margin` and `padding`.

If we increase the `font-size` of our selector, it will increase the `margin` or `padding` with it, and vice versa! It's super useful.

Here is a slide from the course which sums up how I like to use the two:



Flexbox

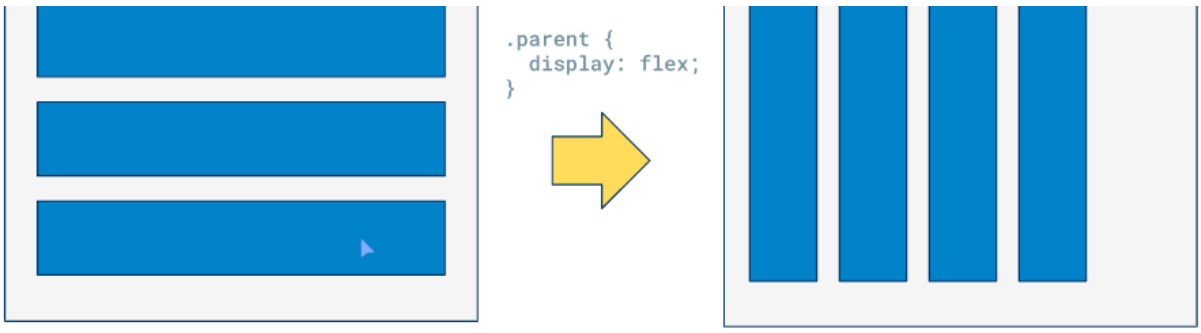
Once we nail down the units and what they are best suited to, I turn the course toward a look at the basics of flexbox, which allows us to create responsive layouts super easily without having to rely on floats or positioning.

All elements, by default, have the `display` property of either `block` or `inline`.

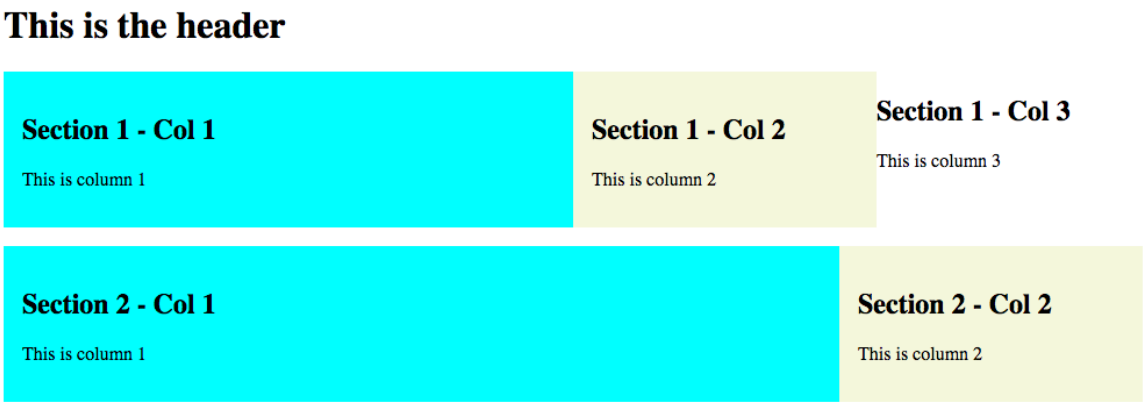
Block elements (`div`, `header`, `footer`, `h1 -> h6`, `p`, etc.) stack on top of each other.

Inline elements (`a`, `strong`, `em`, `span`, etc.) stay within the flow of what is around them.

When we change their display to flex, they start aligning side by side.



Let's see how to design a simple architecture of a typical responsive website with some columns using flexbox. The result would be the following layout:



We will will start off with the following HTML:

```
<body>
  <div class="container">
    <h1>This is the header</h1>
    <div class="columns">
      <div class="col">
        <h2>Section 1 - Col 1</h2>
        <p>This is column 1</p>
      </div>
      <div class="col">
        <h2>Section 1 - Col 2</h2>
        <p>This is column 2</p>
      </div>
      <div class="col">
```

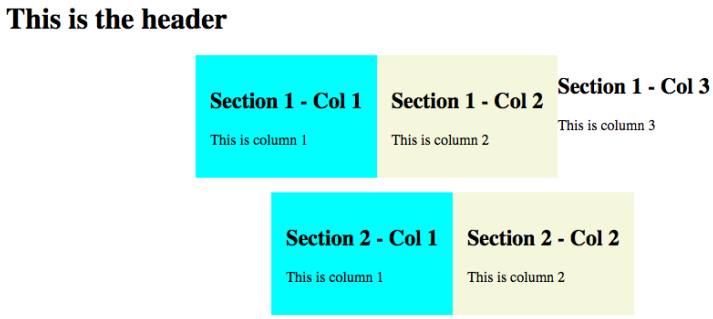
```
    <div class="columns">
      <div class="col">
        <h2>Section 2 - Col 1</h2>
        <p>This is column 1</p>
      </div>
      <div class="col">
        <h2>Section 2 - Col 2</h2>
        <p>This is column 2</p>
      </div>
    </div>
  </div>
</body>
```

And, we can have the following CSS:

```
.container {
  width: 90%;
  max-width: 980px;
  margin: 0 auto;
}

.columns {
  display: flex;
  margin: 1em 0;
}
```

To add background background colors to the columns, we will add couple of modifier classes and assign them to the columns:

```
.col-bg-beige {
  background-color: beige;
  padding: 1em;
}

.col-bg-aqua {
  background-color: aqua;
  padding: 1em;
}
```



In the next step, we want to set the width of columns, we will define the modifier classes and assign the fixed widths inside them.

```
.col-1 {
  width: 25%;
}
.col-2 {
  width: 50%;
}
.col-3 {
  width: 75%;
}
.col-4 {
  width: 100%;
}
```

We will assign these classes to our columns as per the layout needs such as:

If we want our flex columns to align with each other, we can use the following property:

```
.columns {
  display: flex;
  justify-content: center;
}
```

There is a lot more detail about Flexbox basics with code samples and online code editing capability in the Scrimba course ***The Responsive Web Design Bootcamp*****.**

Media Queries

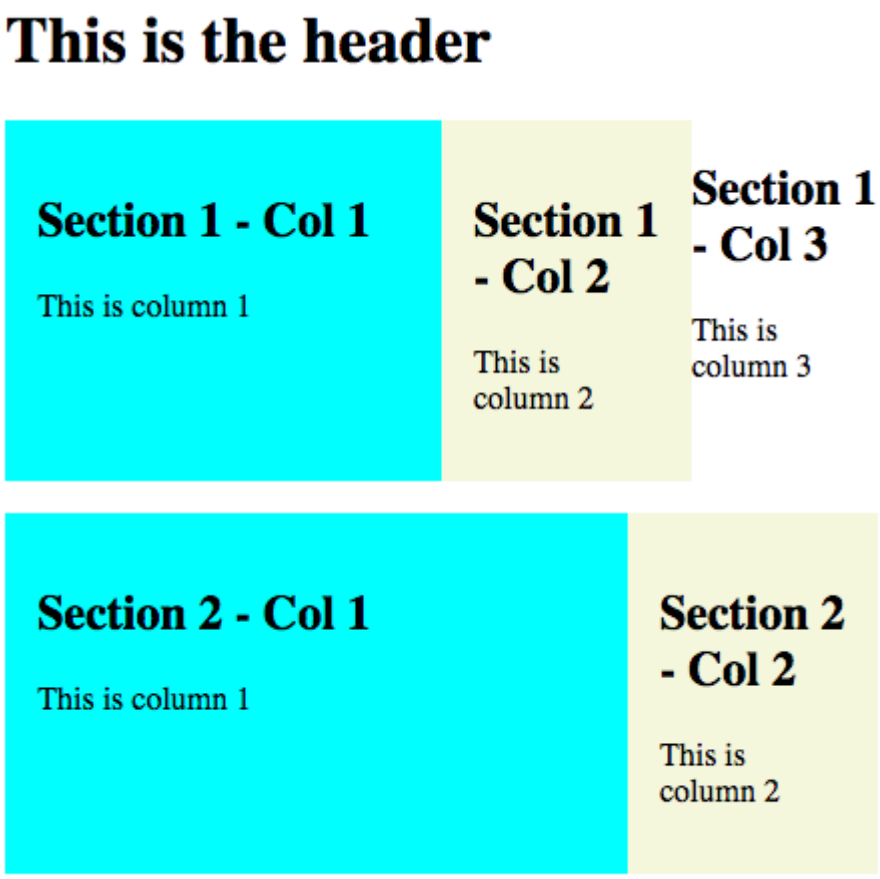
Media queries allow us to declare new CSS rules that only apply in specific situations, such as different types of media (screen, print, speech), and with different media features, such as screen width, orientation, aspect ratio and much more.

The basic syntax for a media query looks like this:

```
@media media-type and (media-features) {...}
```

```
@media screen and (min-width: 480px) {
    body {
        background-color: aqua;
    }
}
```

Going back to our example from above, if we reduce the screen size, we see that some of the columns become way more narrow and we want to fix them by making it fully responsive.



Since we are already using flexbox, we can change the `flex-direction` property to switch the main axis, so instead of creating columns, it will create rows of content.

```
  }
}
```

Designing a responsive navigation

One of my favorite places to take a look at how to apply all of the above into a realistic example is by taking a look at setting up a responsive navigation.

We will start off with writing the HTML for the navigation bar.

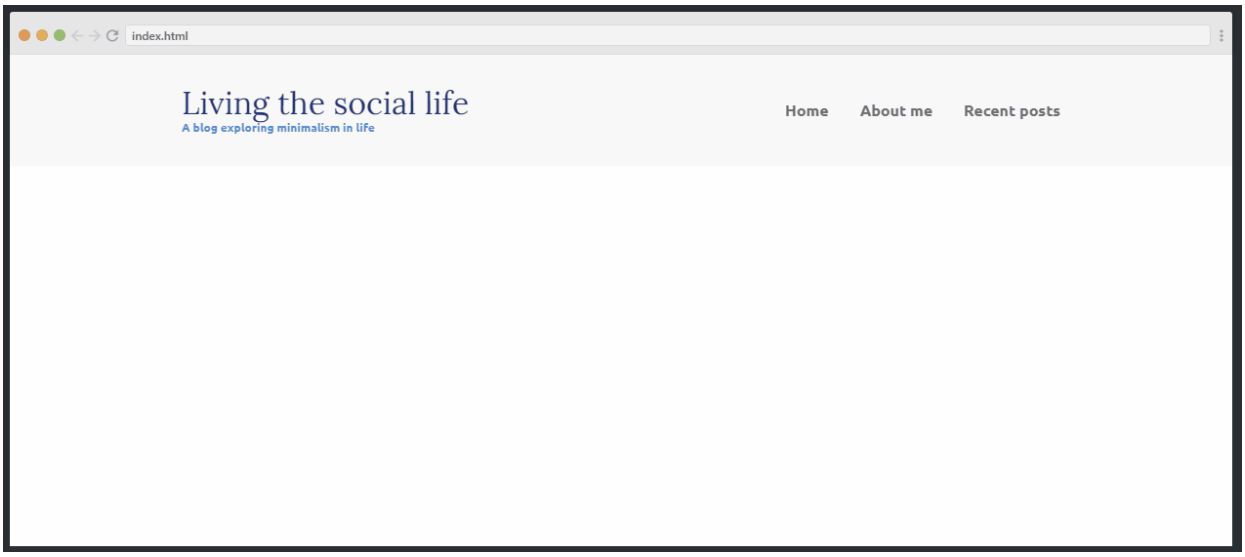
```
<header>
  <div class="container container-nav">
    <div class="site-title">
      <h1>Living the social life</h1>
      <p class="subtitle">A blog exploring minimalism in life</p>
    </div>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
```

```
  </div>
</header>
```

And then we can use flexbox, media queries, a few other little things here and there to start styling it up in our CSS.

Here is the glimpse of some of the CSS (detailed code is available in the course lessons):

```
body {
  margin: 0;
}

.container {
  width: 90%;
  max-width: 900px;
  margin: 0 auto;
}

.container-nav {
  display: flex;
  justify-content: space-between;
}

nav ul {
  display: flex;
  justify-content: center;
  list-style: none;
  margin: 0;
  padding: 0;
}

@media (max-width: 675px) {
  .container-nav {
    flex-direction: column;
  }

  header {
    text-align: center;
  }
}
```

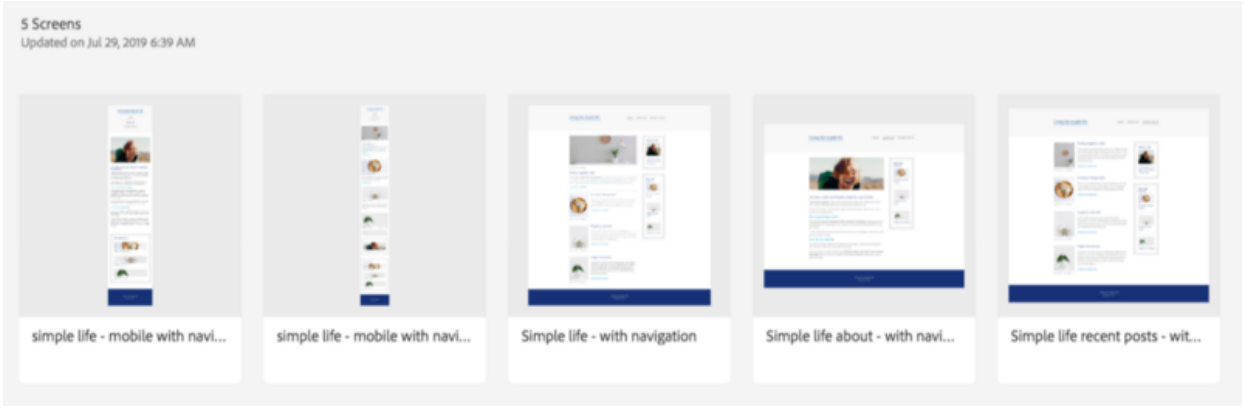


Building out a 3-page site

Examining the structure

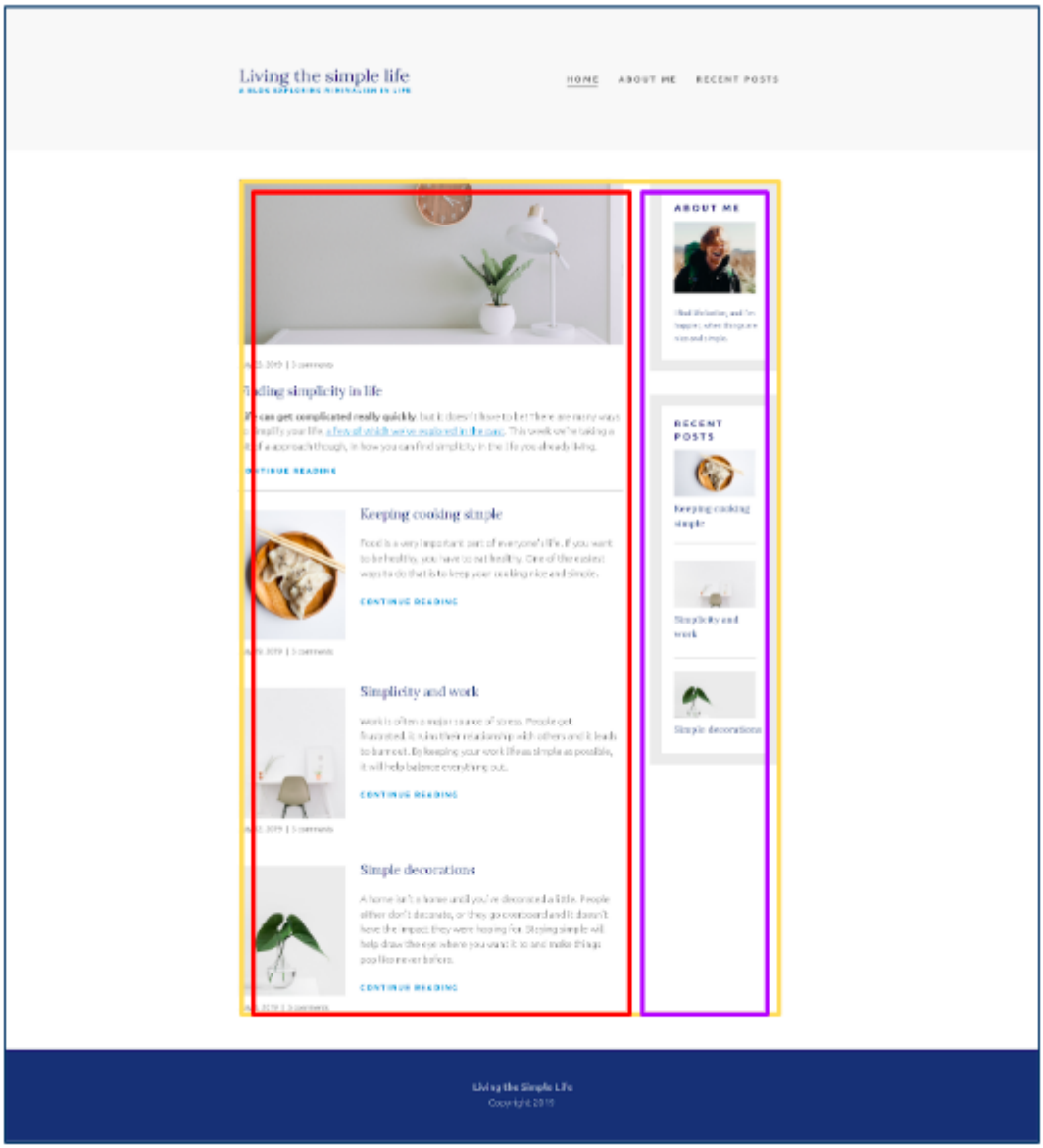
With the navigation wrapped up, I jump into the full-fledge project. This is going to be a 3-page mobile responsive website.

Below you can see the pages we're buildingout, as well as a link to Adobe XD where you can check out the artboards in more detail. In the lessons, I take a look at the similarities across the different pages, and how I start planning for something like this before starting to write any code.



The Home page

As we can see from the image below that there are two main sections of the home page. The left side list of articles with a featured article on top, and the right hand panel with author's info and the recent posts.



We will wrap the whole content in `container` div and first of all create the left hand section which is a list of articles. This is going to be created with the help of `article` tags of the HTML and then we will put content inside them

```
    <article class="article-featured">
      <h2 class="article-title"></h2> <img src="" alt="" class="article-image"
      <p class="article-info"></p>
      <p class="article-body"></p>
      <a href="#" class="article-read-more"></a>
    </article>
    ...
  </main>
</div>
```

Next up we will create the right hand panel with the help of `aside` tag of the HTML.

```
<aside class="sidebar">
  <div class="sidebar-widget">
    <h2 class="widget-title"></h2>
    <img src="#" alt="" class="widget-image">
    <p class="widget-body"></p>
  </div>

    ...
</aside>
```

Before jumping into the layout itself, I like to always start with the typography. It might not be the most exciting part of writing CSS, but when the fonts and font-sizes start to change, it can have a huge impact on the layout. Starting there, then worrying about the layout makes our lives so much easier.

With the typography in place, it's time to start working on the layout itself. While we can start with large screens and then use a media query to redesign things at smaller sizes (like we did with the navigation above), I find it much less work to work the other way around.

Mobile layouts tend to be much simpler, so by starting mobile first, we can create our layouts without too much work.

One the mobile layout is done, then it's time to add in our media query (or queries) and start working on modifying the layout for larger screens.

One thing that's *really* important is not not force yourself to have breakpoints for specific devices. It's all too common to see the same breakpoints used all the time, but really, you should be making layout changes when your layout dictates that it's needed.

I really don't care what device someone is on, or what the size of that device is. What's important is it looks good on all devices. With the amount of different devices and screen sizes today, focus on the layout itself.

In the course itself, I dive into looking at how we can figure out when a layout needs a breakpoint (often because the lines of text are getting too long!), and then we dive into a few fun extras, looking at properties such as

The second and third pages are quite easy to create once the first one is done. By having looked at all the pages before starting, and naming things with classes that can be reused across each page, there are small tweaks to do here and there, but overall things move very fast.

Conclusion

In this article, we examined some core CSS principles to designing responsive websites, from their units to media queries and flexbox. We looked at how we can set it all up for a navgation, and how to break down and analyze a bigger design.

As I mentioned off the top, all of this content is from the *Starting to think responsively* module of the course **The Responsive Web Design Bootcamp**.

The course itself dives much deeper into these topics and many others, from an exploration of the fundamentals of CSS, as well as deep dives into both flexbox and grid, including building out several other projects.

Es la era de *los primeros sitios web móviles* . La buena noticia es que podemos lograr todo lo que necesitamos usando CSS. Ahora hay muchos recursos y herramientas geniales disponibles con los que podemos crear hermosos sitios web compatibles con dispositivos móviles sin la necesidad de marcos o bibliotecas. CSS ha llegado a un lugar *realmente* divertido. ¡Realmente amo CSS, y espero que me acompañes en el curso donde espero poder ayudarte a enamorarte de él también!

kevin powell

Tratando de ayudar a las personas a enamorarse de CSS

---

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) charity organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can make a tax-deductible donation here.

Trending Guides

What is a Framework?

What Do CS Majors Do?

Discord Update Failed

Center an Image in CSS

SQL HAVING

What is OOP?

HTML textarea

NVM for Windows

Aprenda a programar: plan de estudios gratuito de 3000 horas

Python Switch Statement
Python string.replace()
What is a Relational DB?
Split a String in Python
Git List Remote Branches
Git Delete Remote Branch
Software Developer Career
Three Dots Operator in JS
How to Format Dates in JS
2D Array in Java
How to Install NVM
Percentages in Excel
JavaScript Timestamp
Remove Item from Array JS
Dual Boot Windows + Ubuntu
Python Round to 2 Decimals
String to Int in JavaScript
What's the .gitignore File?

nuestra caridad