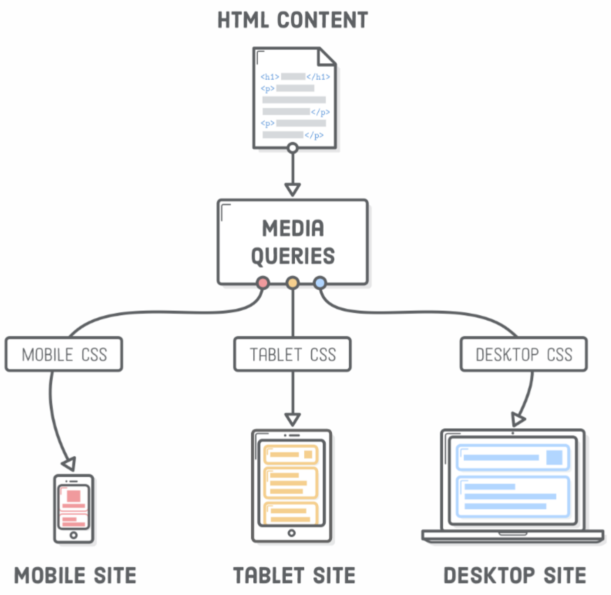
RESPONSIVE DESIGN

Nº 10. OF HTML & CSS IS HARD

A beginner's tutorial for crafting mobile-friendly websites

"Responsive design" refers to the idea that your website should display equally well in everything from widescreen monitors to mobile phones. It's an approach to web design and development that eliminates the distinction between the mobile-friendly version of your website and its desktop counterpart. With responsive design, they're the same thing.

Responsive design is accomplished through CSS "media queries". Think of media queries as a way to conditionally apply CSS rules. They tell the browser that it should ignore or apply certain rules depending on the user's device.



Media queries let us present the same HTML content as distinct CSS layouts. So, instead of maintaining one website for smartphones and an entirely unrelated site for laptops/desktops, we can use the same HTML markup (and web server) for both of them. This means that whenever we add a new article or edit a typo in our HTML, those changes are automatically reflected in both mobile and widescreen layouts. *This* is the reason why we separate content from presentation.

In this chapter, we'll learn how media queries are really just a thin wrapper around the plain old CSS that we've been working with up 'til this point. As we'll soon discover, it's actually pretty easy to implement a responsive layout. (Responsive Images, on the other hand, are an entirely different story).

Media queries let us present the same HTML content as distinct CSS

layouts. So, instead of maintaining one website for smartphones and an entirely unrelated site for laptops/desktops, we can use the same HTML markup (and web server) for both of them. This means that whenever we add a new article or edit a typo in our HTML, those changes are automatically reflected in both mobile and widescreen layouts. *This* is the reason why we separate content from presentation.

In this chapter, we'll learn how media queries are really just a thin wrapper around the plain old CSS that we've been working with up 'til this point. As we'll soon discover, it's actually pretty easy to implement a responsive layout. (Responsive Images, on the other hand, are an entirely different story).

SETUP

add a new article or edit a typo in our HTML, those changes are automatically reflected in both mobile and widescreen layouts. *This* is the reason why we separate content from presentation.

In this chapter, we'll learn how media queries are really just a thin wrapper around the plain old CSS that we've been working with up 'til this point. As we'll soon discover, it's actually pretty easy to implement a responsive layout. (Responsive Images, on the other hand, are an entirely different story).

SETUP

Create a new project called `responsive-design` and a new file called `responsive.html`. It's the emptiest web page that we've seen in awhile, but it'll help us demonstrate something very important in the next section:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Responsive Design</title>
    <link rel='stylesheet' href='styles.css'/>
  </head>
  <body>
    <!-- There's nothing here! -->
  </body>
</html>
```

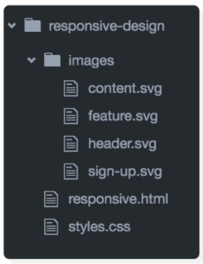
SETUP

Create a new project called `responsive-design` and a new file called `responsive.html`. It's the emptiest web page that we've seen in awhile, but it'll help us demonstrate something very important in the next section:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Responsive Design</title>
    <link rel='stylesheet' href='styles.css'/>
  </head>
  <body>
    <!-- There's nothing here! -->
  </body>
```

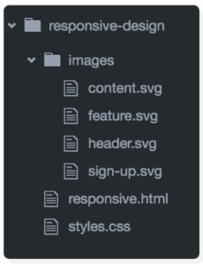
```
</html>
```

You'll also need to download some images for later in the chapter. Unzip everything into the same folder as `responsive.html`, keeping the parent `images` folder. Your project should look like this before moving on:



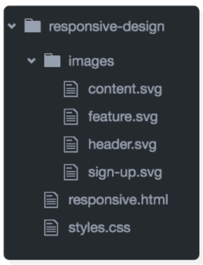
```
responsive-design
  images
    content.svg
    feature.svg
    header.svg
    sign-up.svg
  responsive.html
  styles.css
```

You'll also need to download some images for later in the chapter. Unzip everything into the same folder as `responsive.html`, keeping the parent `images` folder. Your project should look like this before moving on:



```
responsive-design
  images
    content.svg
    feature.svg
    header.svg
    sign-up.svg
  responsive.html
  styles.css
```

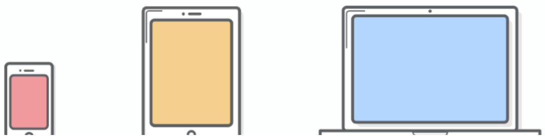
`images` folder. Your project should look like this before moving on:



```
responsive-design
  images
    content.svg
    feature.svg
    header.svg
    sign-up.svg
  responsive.html
  styles.css
```

CSS MEDIA QUERIES

We'll start small by simply updating the background color on the `<body>` element based on the device width. This is a good way to make sure our media queries are actually working before getting into complicated layouts.



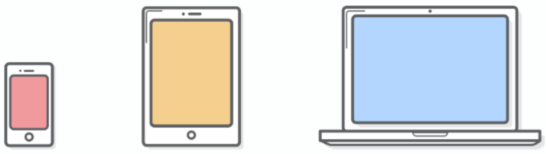
CSS MEDIA QUERIES

We'll start small by simply updating the background color on the `<body>` element based on the device width. This is a good way to make sure our media queries are actually working before getting into complicated layouts.

Let's differentiate between narrow, medium, and wide layouts by creating a new `styles.css` stylesheet and adding the following:

CSS MEDIA QUERIES

We'll start small by simply updating the background color on the `<body>` element based on the device width. This is a good way to make sure our media queries are actually working before getting into complicated layouts.



Let's differentiate between narrow, medium, and wide layouts by creating a new `styles.css` stylesheet and adding the following:

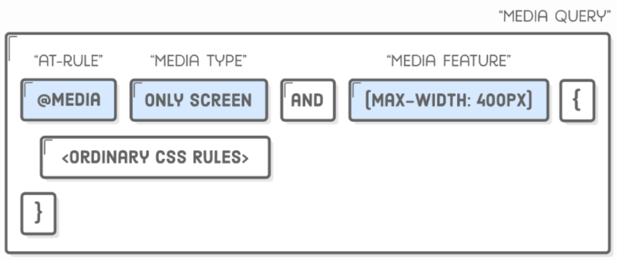
```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

/* Mobile Styles */
@media only screen and (max-width: 400px) {
  body {
    background-color: #F09A9D; /* Red */
  }
}

/* Tablet Styles */
@media only screen and (min-width: 401px) and (max-width: 960px) {
  body {
    background-color: #F5CF8E; /* Yellow */
  }
}
```

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

/* Mobile Styles */
@media only screen and (max-width: 400px) {
  body {
    background-color: #F09A9D; /* Red */
  }
}
```

```
/* Tablet Styles */
@media only screen and (min-width: 401px) and (max-width: 960px) {
  body {
    background-color: #F5CF8E; /* Yellow */
  }
}

/* Desktop Styles */
@media only screen and (min-width: 961px) {
  body {
  box-sizing: border-box;
}

/* Mobile Styles */
@media only screen and (max-width: 400px) {
  body {
    background-color: #F09A9D; /* Red */
  }
}

/* Tablet Styles */
@media only screen and (min-width: 401px) and (max-width: 960px) {
  body {
    background-color: #F5CF8E; /* Yellow */
  }
}

/* Desktop Styles */
@media only screen and (min-width: 961px) {
  body {
    background-color: #B2D6FF; /* Blue */
  }
}
```

When you resize your browser, you should see three different background colors: blue when it's greater than `960px` wide, yellow when it's between `401px` and `960px`, and red when it's less than `400px`.

Media queries always begin with the `@media` "at-rule" followed by some kind of conditional statement, and then some curly braces. Inside the curly braces, you put a bunch of ordinary CSS rules. The browser only pays attention to those rules if the condition is met.



The `only screen` "media type" means that the contained styles should only be applied to devices with screens (opposed to printed documents, like when you hit **Cmd+P** in a browser). The `min-width` and `max-width` parts are called "media features", and they specify the device dimensions you're targeting.

The above media queries are by far the most common ones you'll encounter, but there are a lot of other conditions you can check for, including whether the device is in portrait or landscape mode, the resolution of its screen, and whether it has a mouse or not.

The `only screen` "media type" means that the contained styles should only be applied to devices with screens (opposed to printed documents, like when you hit **Cmd+P** in a browser). The `min-width` and `max-width` parts are called "media features", and they specify the device dimensions you're targeting.

The above media queries are by far the most common ones you'll encounter, but there are a lot of other conditions you can check for, including whether the device is in portrait or landscape mode, the resolution of its screen, and whether it has a mouse or not.

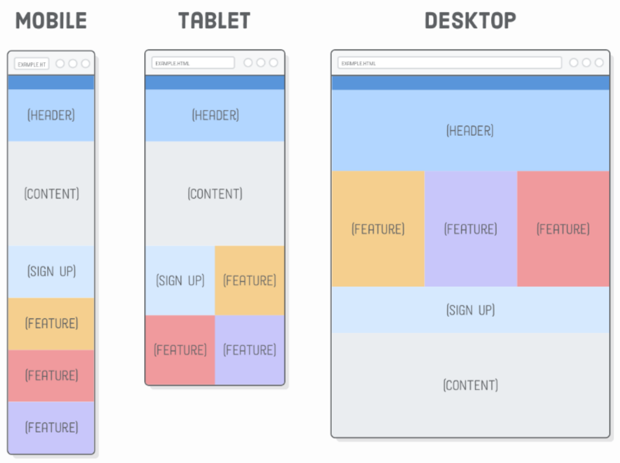
A FEW NOTES ON DESIGN

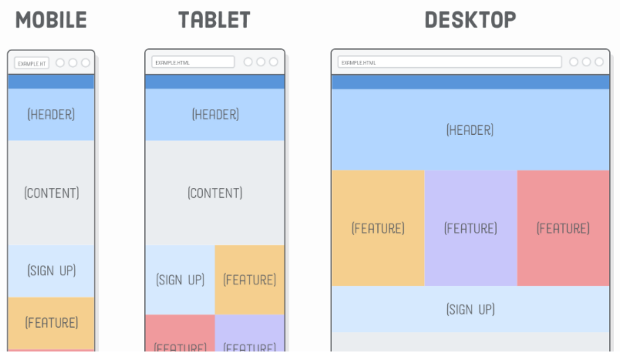
Ok, so `@media` is how we define different layouts for specific device widths, but what layouts are we actually trying to implement? The example web page for this chapter is going to look something like this:

but there are a lot of other conditions you can check for, including whether the device is in portrait or landscape mode, the resolution of its screen, and whether it has a mouse or not.

A FEW NOTES ON DESIGN

Ok, so `@media` is how we define different layouts for specific device widths, but what layouts are we actually trying to implement? The example web page for this chapter is going to look something like this:

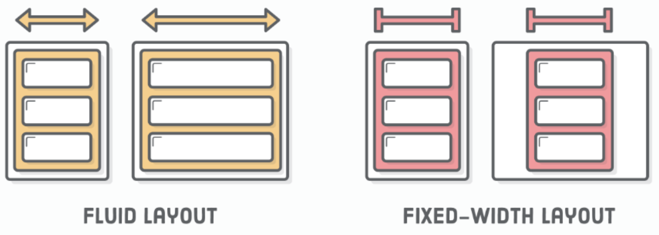




In the real world, it's up to your web designer to supply you with these kinds of mockups. Your job as a developer is to implement the individual layouts using media queries to separate out the various CSS rules that apply to each one.

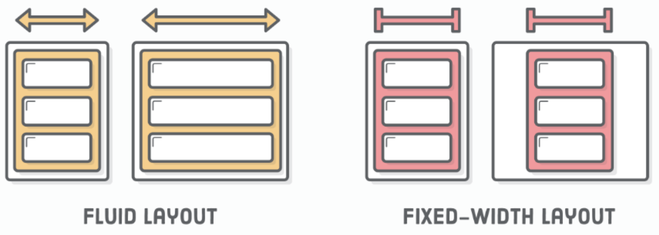
There's a few well defined patterns for how a desktop layout collapses into a mobile layout (we're using "layout shifter"). A lot of these decisions are in the realm of design, which is outside the scope of this code-oriented tutorial; however, there are two concepts that you must understand as a developer:

- A "fluid" layout is one that stretches and shrinks to fill the width of the screen, just like the flexible boxes we covered a few chapters ago.
- A "fixed-width" layout is the opposite: it has the same width regardless of the screen dimensions (we created one of these in the CSS Selectors chapter).



In our example web page, the mobile and tablet versions are fluid, and the desktop version is fixed-width.

- A "fluid" layout is one that stretches and shrinks to fill the width of the screen, just like the flexible boxes we covered a few chapters ago.
- A "fixed-width" layout is the opposite: it has the same width regardless of the screen dimensions (we created one of these in the CSS Selectors chapter).



In our example web page, the mobile and tablet versions are fluid, and the desktop version is fixed-width.

CHOOSING BREAKPOINTS

Most of those responsive design patterns have similar behavior, using fluid layouts for mobile/tablet devices and fixed-width layouts for wider screens. There's a reason for this.

Fluid layouts let us target a *range* of screen widths instead of specific mobile devices. This is very important for web designers. When they set out to create a mobile layout, they aren't trying to make something that looks good on an iPhone 6s, Galaxy S7, or iPad mini—they're designing a fluid layout that looks good *anywhere* between 300 pixels and 500 pixels (or

FLUID LAYOUT FIXED-WIDTH LAYOUT

In our example web page, the mobile and tablet versions are fluid, and the desktop version is fixed-width.

CHOOSING BREAKPOINTS

Most of those responsive design patterns have similar behavior, using fluid layouts for mobile/tablet devices and fixed-width layouts for wider screens. There's a reason for this.

Fluid layouts let us target a *range* of screen widths instead of specific mobile devices. This is very important for web designers. When they set out to create a mobile layout, they aren't trying to make something that looks good on an iPhone 6s, Galaxy S7, or iPad mini—they're designing a fluid layout that looks good *anywhere* between 300 pixels and 500 pixels (or whatever).

In other words, the exact pixel values for the `min-width` and `max-width` parameters in a media query (collectively known as the "breakpoints" for a responsive website) don't actually matter. Our website doesn't care about the specific device the user is on. All it needs to know is that it should display a layout that looks pretty at 400 pixels wide (or whatever).

Most of those responsive design patterns have similar behavior, using fluid layouts for mobile/tablet devices and fixed-width layouts for wider screens. There's a reason for this.

Fluid layouts let us target a *range* of screen widths instead of specific mobile devices. This is very important for web designers. When they set out to create a mobile layout, they aren't trying to make something that looks good on an iPhone 6s, Galaxy S7, or iPad mini—they're designing a fluid layout that looks good *anywhere* between 300 pixels and 500 pixels (or whatever).

In other words, the exact pixel values for the `min-width` and `max-width` parameters in a media query (collectively known as the "breakpoints" for a responsive website) don't actually matter. Our website doesn't care about the specific device the user is on. All it needs to know is that it should display a layout that looks pretty at 400 pixels wide (or whatever).

There's a reason for this.

Fluid layouts let us target a *range* of screen widths instead of specific mobile devices. This is very important for web designers. When they set out to create a mobile layout, they aren't trying to make something that looks good on an iPhone 6s, Galaxy S7, or iPad mini—they're designing a fluid layout that looks good *anywhere* between 300 pixels and 500 pixels (or whatever).

In other words, the exact pixel values for the `min-width` and `max-width` parameters in a media query (collectively known as the "breakpoints" for a responsive website) don't actually matter. Our website doesn't care about the specific device the user is on. All it needs to know is that it should display a layout that looks pretty at 400 pixels wide (or whatever).

There's a reason for this.

Fluid layouts let us target a *range* of screen widths instead of specific mobile devices. This is very important for web designers. When they set out to create a mobile layout, they aren't trying to make something that looks good on an iPhone 6s, Galaxy S7, or iPad mini—they're designing a fluid layout that looks good *anywhere* between 300 pixels and 500 pixels (or whatever).

In other words, the exact pixel values for the `min-width` and `max-width` parameters in a media query (collectively known as the "breakpoints" for a responsive website) don't actually matter. Our website doesn't care about the specific device the user is on. All it needs to know is that it should display a layout that looks pretty at 400 pixels wide (or whatever).

mobile devices. This is very important for web designers. When they set out to create a mobile layout, they aren't trying to make something that looks good on an iPhone 6s, Galaxy S7, or iPad mini—they're designing a fluid layout that looks good *anywhere* between 300 pixels and 500 pixels (or whatever).

In other words, the exact pixel values for the `min-width` and `max-width` parameters in a media query (collectively known as the "breakpoints" for a responsive website) don't actually matter. Our website doesn't care about the specific device the user is on. All it needs to know is that it should display a layout that looks pretty at 400 pixels wide (or whatever).

MOBILE-FIRST DEVELOPMENT

Let's dive right into implementing the above screenshots. It's always a good idea to start with the mobile layout and work your way up to the desktop version. Desktop layouts are typically more complex than their mobile counterparts, and this "mobile-first" approach maximizes the amount of CSS that you can reuse across your layouts.

First, we need to fill in `responsive.html`'s `<body>` element with some empty boxes. Each box has an image in it so we can tell them apart a little bit easier.

MOBILE-FIRST DEVELOPMENT

Let's dive right into implementing the above screenshots. It's always a good idea to start with the mobile layout and work your way up to the desktop version. Desktop layouts are typically more complex than their mobile counterparts, and this "mobile-first" approach maximizes the amount of CSS that you can reuse across your layouts.

First, we need to fill in `responsive.html`'s `<body>` element with some empty boxes. Each box has an image in it so we can tell them apart a little bit easier.

```html
<div class='page'>
  <div class='section menu'></div>
  <div class='section header'>
    <img src='images/header.svg'/>
  </div>
  <div class='section content'>
    <img src='images/content.svg'/>
  </div>
  <div class='section sign-up'>
    <img src='images/sign-up.svg'/>
  </div>
  <div class='section feature-1'>
    <img src='images/feature.svg'/>
  </div>
  <div class='section feature-2'>
    <img src='images/feature.svg'/>
  </div>
  <div class='section feature-3'>
    <img src='images/feature.svg'/>
  </div>
</div>
```

And here’s our base styles, which should apply to *all* layouts (mobile, tablet, and desktop). Make sure to add these above the `@media` rules we created earlier and below the universal selector rule that resets our margins and padding:

```
.page {
  display: flex;
  flex-wrap: wrap;
}

.section {
  width: 100%;
  height: 300px;
}

.menu {
  background-color: #5995DA;
  height: 80px;
}

.header {
  background-color: #B2D6FF;
}

.content {
  background-color: #EAEDF0;
  height: 600px;
}

.sign-up {
  background-color: #D6E9FE;
}

.feature-1 {
  background-color: #F5CF8E;
}

.feature-2 {
  background-color: #F09A9D;
```