Lesson | Thursday

Introduction to Programming (/introduction-to-programming) / Git, HTML and CSS (/introduction-to-programming/git-html-and-css) / CSS Media Queries and Responsive Design

Text | Cheat sheet

You may notice some of your frequently-visited websites look a little different on your phone than they do in the browser on a computer. Altering the appearance of a website based on the device it is being viewed on is an important feature of CSS that ensures a quality viewing experience on any device. In this lesson, we'll learn more about adapting your site for different screen sizes using media queries.

Responsive Design

Coding your site to look different depending on the size of screen it is being viewed on is known as **responsive web design**. In the realm of responsive web design, the screen being used to view a site is commonly referred to as a **viewport**.

But what does responsive design look like? How should a site look on a small screen, versus a larger screen? Before we begin, check out the following two websites to view some examples of

responsive design. Each site depicts the layouts of many different websites, and the way their styles appear differently depending on the user's viewport size:

- Responsive Design (https://responsivedesign.is/examples)
- mediaqueri.es (http://mediaqueri.es/)

Making a Site Responsive

But how do we make our own sites responsive? How does it know when to use styles meant for smaller viewports, and when to apply styles meant for larger ones? Thankfully, this is actually easier than you might anticipate.

To make sites responsive we can use CSS media queries. A **media query** is a block of CSS that is applied *only* when certain conditions about the user's viewport are true. For instance, CSS packaged in a media query that defines a maximum width of 500px would only be applied when the user's viewport width is below 500px.

Media Query Practice

Let's walk through creating media queries together. This will allow us to both understand their construction, and see what they look like in action.

We'll create a project directory named `media-query-practice`. Within it, we'll need a `.html` file named `media-query-site.html`, and another directory named `css`. In the `css` directory we'll create a file called `styles.css`.

The project structure should look like this:

```
media-query-practice
├── css
│   └── styles.css
└── media-query-site.html
```

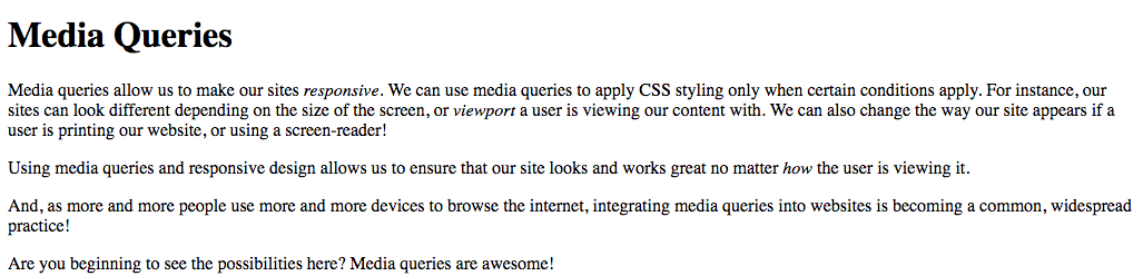
Make sure to initialize a Git repository in the root directory so you'll be ready to start making commits.

Next, let's add some simple HTML:

media-query-site.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <link href="css/styles.css" rel="stylesheet" type="text/c
ss">
  <title>Media Queries</title>
</head>
<body>
  <h1>Media Queries</h1>
  <div class="column">
    <p>Media queries allow us to make our sites <em>respons
ive</em>. We can use media queries to apply CSS styling onl
y when certain conditions apply. For instance, our sites ca
n look different depending on the size of the screen, or <e
m>viewport</em> a user is viewing our content with. We can
also change the way our site appears if a user is printing
our website, or using a screen-reader! </p>
  </div>
  <div class="column">
    <p>Using media queries and responsive design allows us
to ensure that our site looks and works great no matter <em
>how</em> the user is viewing it. And, as more and more peo
ple use more and more devices to browse the internet, integ
rating media queries into websites is becoming a common, wi
despread practice!</p>
  </div>
  <div class="column">
    <p>Are you beginning to see the possibilities here? Med
ia queries are awesome!</p>
  </div>
</body>
</html>
```

If we load our `media-query-site.html` file in the browser, it should currently look like this:



Media Queries

Media queries allow us to make our sites *responsive*. We can use media queries to apply CSS styling only when certain conditions apply. For instance, our sites can look different depending on the size of the screen, or *viewport* a user is viewing our content with. We can also change the way our site appears if a user is printing our website, or using a screen-reader!

Using media queries and responsive design allows us to ensure that our site looks and works great no matter *how* the user is viewing it.

And, as more and more people use more and more devices to browse the internet, integrating media queries into websites is becoming a common, widespread practice!

Are you beginning to see the possibilities here? Media queries are awesome!

Anatomy of a Media Query

Now that we have a basic site in place, let's add media queries. We'll begin by building one from scratch together.

Basic Structure

Media queries are located in CSS stylesheets. Because CSS cascades, they should be located at the *bottom* of the file, to prevent other style rules from overriding them.

They begin with `@media`, and contain a set of parenthesis and curly braces, like this:

styles.css

```
@media () {

}
```

- The `@media` portion instructs the browser that this is a media query.
- The parenthesis will eventually define *when* the CSS in this query should be applied to our site. We'll get to this in a moment.
- The curly braces will eventually contain the CSS that will be applied when the conditions we define are true.

Media Type

Media queries often specify something called a media type. As the name implies, a **media type** refers to the type of media-viewing device the user is viewing our site with. We have the following options to choose from:

- `all` : Refers to any and all devices.
- `print` : Refers to viewing our site in "print preview" mode. For instance, if we had a website with a dark-colored background that users may want to print, we could use a `print` media query that removes the dark-colored background for `print` media types. This would both make our website's information more legible in a printed format, and save our users printer ink.
- `screen` : Refers to color screens.
- `speech` : Refers to special screen readers that assist individuals with impaired eyesight.

Additionally, note that if you do not explicitly specify a type, your media query will default to `all` .

We can add the media type to our query like this:

styles.css

```
@media screen () {

}
```

By including the media type `screen` we're instructing our site to only apply the CSS that this media query will eventually contain to the site if a user is viewing it through a screen.

Media Features

Next, our media query requires we include something called a media feature. **Media features** are specific properties and details about the manner the user is viewing content. The most commonly-used media features are:

- `height` : Describes the height of the viewport, in pixels. This media feature can also have a `min` or `max` prefix added onto it. That is, we can say `max-height` to specify the maximum height a media query's CSS should apply to. Or `min-height` to define a minimum height.
- `width` : Describes the width of the viewport, in pixels. Like `height` , it may also have a `min` or `max` prefix added onto it.
- `orientation` : Indicates whether the viewport is `landscape` (wider than it is tall) or `portrait` (taller than it is wide).
- While not required for this course, you can learn about the additional, less common features in the MDN Documentation on Media Queries (https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries#Media_features).

Let's use the `width` media feature in our media query. We'll also include the `max` prefix, in order to specify a maximum width:

styles.css

```
@media screen and (max-width: 768px) {

}
```

We've done a couple things here:

- First, we add the word `and` between our `screen` media type, and the parenthesis containing our new media feature. When using both a media type *and* a media feature this is required.

- Then, we include `max-width: 768px` in our parenthesis.
    - `width` is our media feature.
    - The `max` prefix specifies that the CSS we will eventually include in this media query should only be applied to viewports with a maximum width of `768px`.
- `768px` simply refers to size (in pixels) we'd like to define as the maximum. When defining pixel sizes in CSS, the `px` suffix is required.

Breakpoints

So, this means the CSS we will eventually include in this media query will only be applied when the user's viewport is *under* the maximum width of 768 pixels. This is known as a breakpoint.

A **breakpoint** is the point at which a media query's condition becomes true. For instance, because our media query has a max-width of 768px, it will apply its styles only when the viewport is 768px or less wide. 768px is therefore the breakpoint, because it is the point at which the query will be "activated".

Media Query CSS Rules

We can now include CSS within its opening and closing curly braces. The CSS in a media query is exactly like CSS we've been using so far. We can include any valid CSS in a media query. The only difference is that it will *only* be applied when the conditions defined by our media query are met.

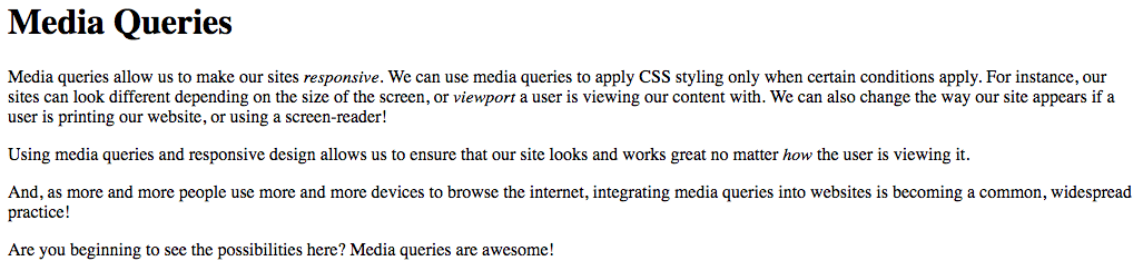
Let's add some basic, yet noticeable styles, so we can readily see when they're being applied:

styles.css

```
@media screen and (max-width: 768px) {
    body {
        background-color: black;
        color: white;
    }
}
```

Here, we're saying that if the user is viewing our site on a `screen`, whose viewport is no larger than 768 pixels, the background will be black and the font will be white.

Let's load our site into the browser to see this in action! If we refresh the page, we can see it still looks the same:



Media Queries

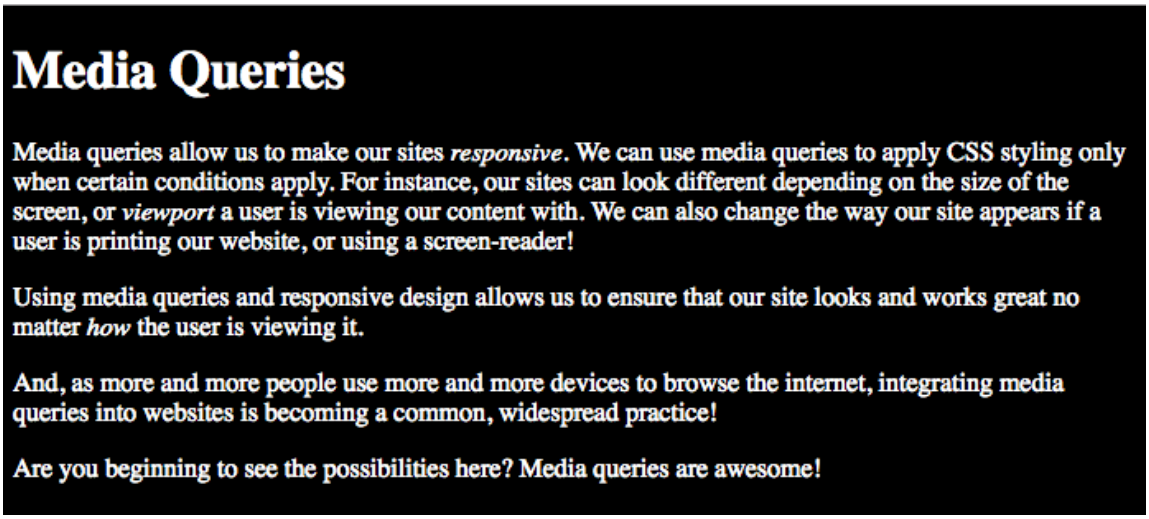
Media queries allow us to make our sites *responsive*. We can use media queries to apply CSS styling only when certain conditions apply. For instance, our sites can look different depending on the size of the screen, or *viewport* a user is viewing our content with. We can also change the way our site appears if a user is printing our website, or using a screen-reader!

Using media queries and responsive design allows us to ensure that our site looks and works great no matter *how* the user is viewing it.

And, as more and more people use more and more devices to browse the internet, integrating media queries into websites is becoming a common, widespread practice!

Are you beginning to see the possibilities here? Media queries are awesome!

However, slowly resize the browser window to make it narrower. As soon as the browser window becomes narrower than 768 pixels the media query is activated; our background turns black, and our text turns white!



Isn't this pretty cool?

Multiple Media Queries

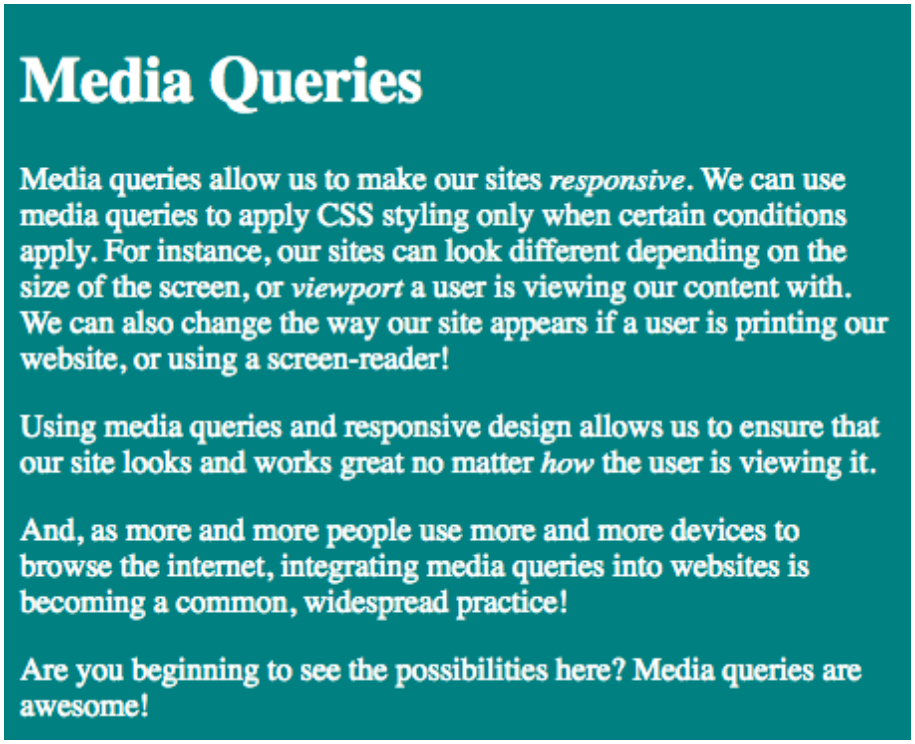
We can use multiple media queries at once to address a wide variety of potential viewport sizes and media types. Let's add another query to our site:

styles.css

```
@media screen and (max-width: 768px) {
  body {
    background-color: black;
    color: white;
  }
}

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

If we refresh our page, we can see it still has a white background if its width is over 768 pixels, and a black background between 480 pixels and 768 pixels. Now, thanks to our second query, our background is teal if the site is fewer than 480 pixels wide!



Notice here that the font color is *still* white after the background turns teal, even though the second media query did not specify a font color. This is because the viewport width is *still* less than 768 pixels; so unless we override the `color` property with a new property, it will continually remain white.

Sometimes multiple media queries can apply at once. For instance, in our media queries above, if a viewport is under 480 pixels *both* media queries will apply because a size under 480 pixels is under *both* the maximum width value of 768 *and* the maximum value of 480.

When multiple media queries apply, the most-recently applied media query's CSS will override the other media queries' CSS if they contain the same selectors and properties. For instance, in the

example above `background-color` defined in the first media query is being overridden by the `background-color` property in our second media query when the viewport is 480 pixels or less in width.

Defining Viewport Ranges

We can also apply both minimum *and* maximum values to a media query. This can especially come in handy if we *don't* want to worry about multiple media queries applying at once.

We could add a `min-width` feature to one of our existing queries like this:

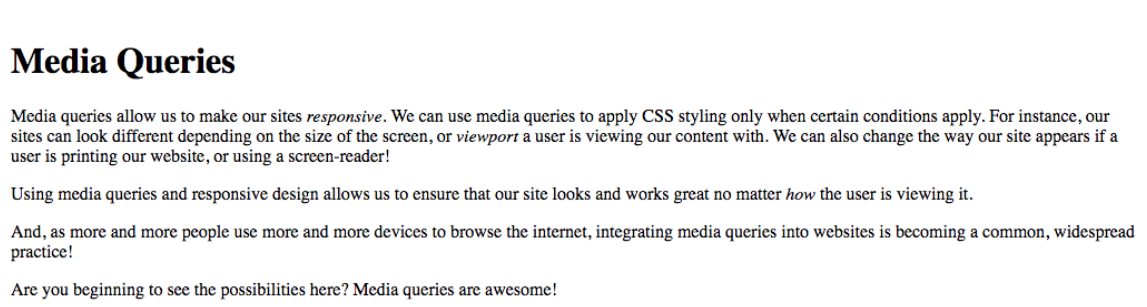
styles.css

```
@media screen and (max-width: 768px) and (min-width: 600px)
{
  body {
    background-color: black;
    color: white;
  }
}

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

Notice that there is another `and` between the two media features in parenthesis, just like the `and` between our media type and first media feature. Now, this first query instructs our site to apply a black background and white text when the viewport is between 768 and 600 pixels wide.

Now, our site begins with a white background, and no styling applied:



Media Queries

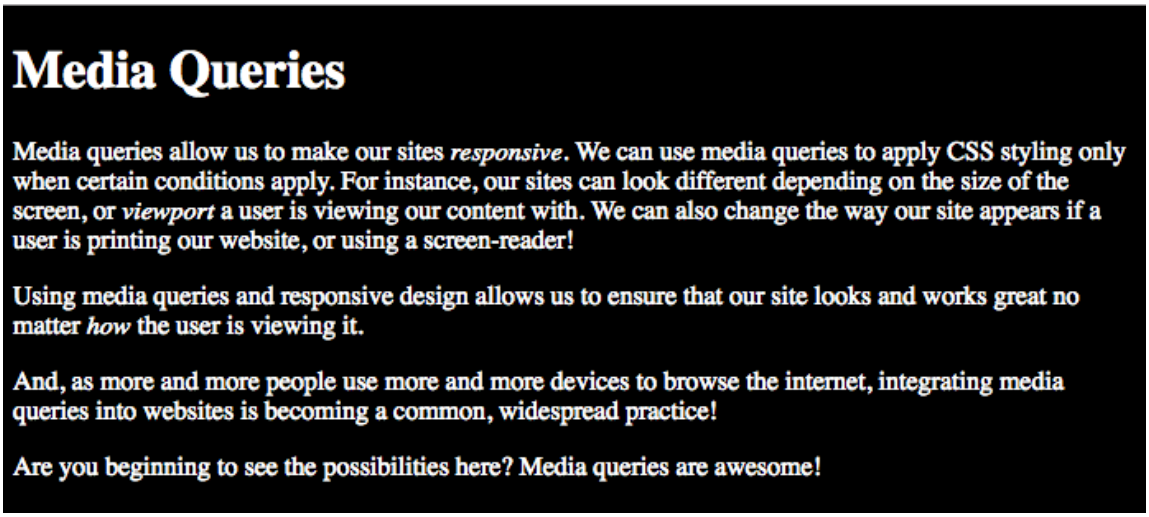
Media queries allow us to make our sites *responsive*. We can use media queries to apply CSS styling only when certain conditions apply. For instance, our sites can look different depending on the size of the screen, or *viewport* a user is viewing our content with. We can also change the way our site appears if a user is printing our website, or using a screen-reader!

Using media queries and responsive design allows us to ensure that our site looks and works great no matter *how* the user is viewing it.

And, as more and more people use more and more devices to browse the internet, integrating media queries into websites is becoming a common, widespread practice!

Are you beginning to see the possibilities here? Media queries are awesome!

As we slowly reduce the width of its browser window, its background turns black, and text turns white at 768px:



Media Queries

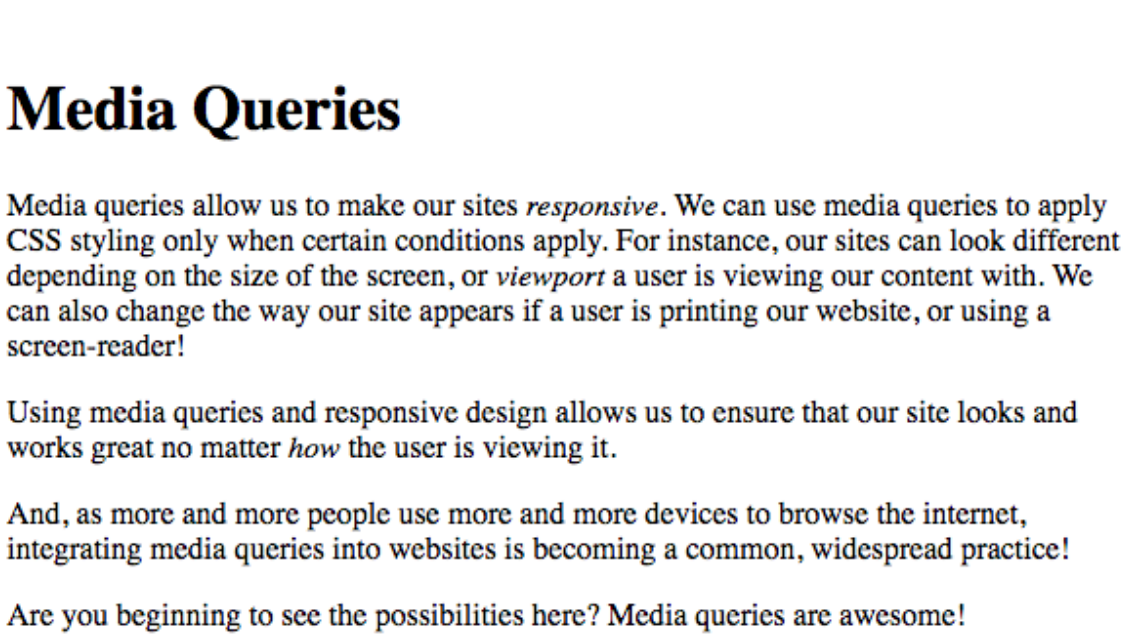
Media queries allow us to make our sites *responsive*. We can use media queries to apply CSS styling only when certain conditions apply. For instance, our sites can look different depending on the size of the screen, or *viewport* a user is viewing our content with. We can also change the way our site appears if a user is printing our website, or using a screen-reader!

Using media queries and responsive design allows us to ensure that our site looks and works great no matter *how* the user is viewing it.

And, as more and more people use more and more devices to browse the internet, integrating media queries into websites is becoming a common, widespread practice!

Are you beginning to see the possibilities here? Media queries are awesome!

But now, once its width is smaller than 600px (but below the 480 pixel width that will activate another media query), it reverts back to white:



Media Queries

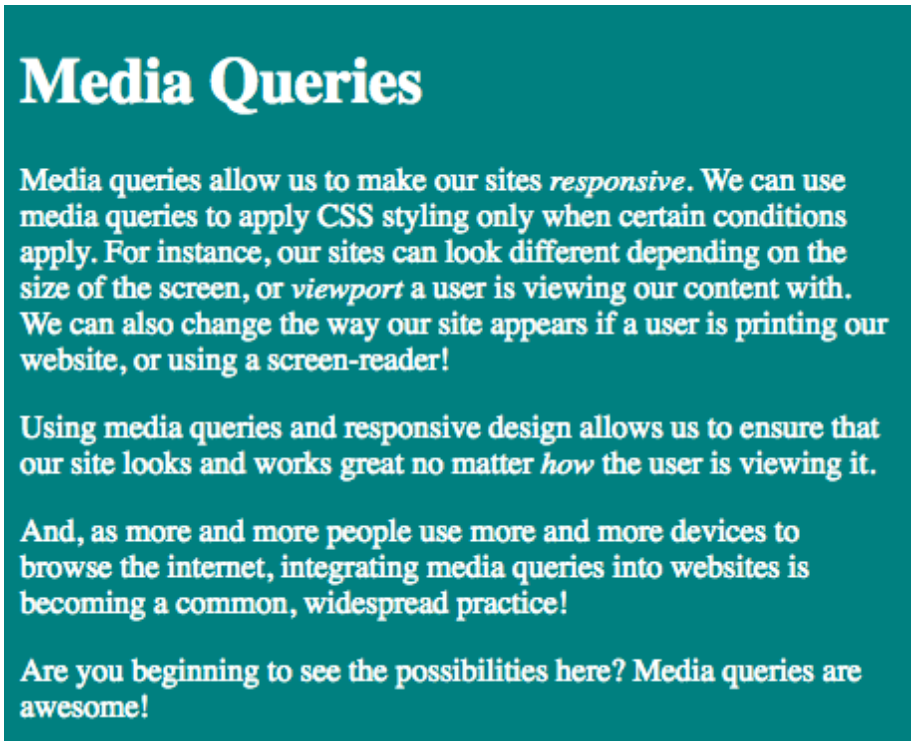
Media queries allow us to make our sites *responsive*. We can use media queries to apply CSS styling only when certain conditions apply. For instance, our sites can look different depending on the size of the screen, or *viewport* a user is viewing our content with. We can also change the way our site appears if a user is printing our website, or using a screen-reader!

Using media queries and responsive design allows us to ensure that our site looks and works great no matter *how* the user is viewing it.

And, as more and more people use more and more devices to browse the internet, integrating media queries into websites is becoming a common, widespread practice!

Are you beginning to see the possibilities here? Media queries are awesome!

Yet, once it reaches 480 pixels or smaller, it still turns teal:



Media Queries

Media queries allow us to make our sites *responsive*. We can use media queries to apply CSS styling only when certain conditions apply. For instance, our sites can look different depending on the size of the screen, or *viewport* a user is viewing our content with. We can also change the way our site appears if a user is printing our website, or using a screen-reader!

Using media queries and responsive design allows us to ensure that our site looks and works great no matter *how* the user is viewing it.

And, as more and more people use more and more devices to browse the internet, integrating media queries into websites is becoming a common, widespread practice!

Are you beginning to see the possibilities here? Media queries are awesome!

More CSS in Media Queries

Also, remember that you can define *any* CSS in a media query; not just background colors! For instance, we learned how to create columns in a previous lesson

(https://www.learnhowtoprogram.com/introduction-to-programming/git-html-and-css/using-floats). Oftentimes, sites will display text in multiple columns on larger screens, then condense it into a single column for easier reading on smaller devices as seen in this example (http://mediaqueri.es/dbp/).

We can do this too! Let's create another media query. This time, we'll use the `min` prefix on the `width` feature. We'll specify that any viewport *above* 768px should float our text into columns. This means that any viewport *below* the size of this breakpoint will display text in a single column:

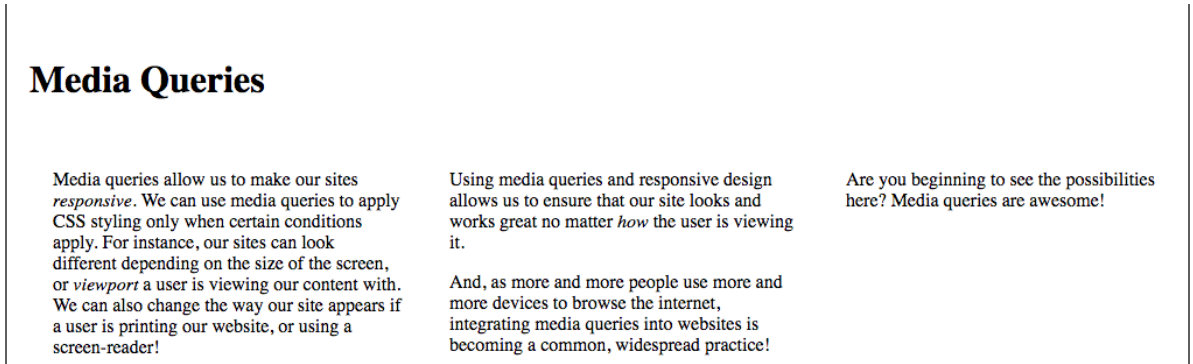
styles.css

```
@media screen and (min-width: 768px) {
  .column {
    width: 300px;
    float: left;
    padding: 20px;
  }
}

@media screen and (max-width: 768px) and (min-width: 600px)
{
  body {
    background-color: black;
    color: white;
  }
}

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

And look! Larger screen sizes see our text in columns:



However, if we resize to a smaller viewport, text will be condensed into a single column.

Conclusion

As you can see, media queries are pretty powerful! Based on the manner in which a user views our site, we can dynamically change what it looks like to appear best. With the increasing number of individuals accessing content on handheld devices and tablets, media queries and responsive design are becoming ever more important.

As you create websites and applications throughout this course, begin implementing media queries to ensure your content looks great on *all* devices.

Previous (/introduction-to-programming/git-html-and-css/video-recap-core-css-concepts-and-debugging)
Next (/introduction-to-programming/git-html-and-css/practice-css-media-queries-and-responsive-design)

Lesson 56 of 64
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)