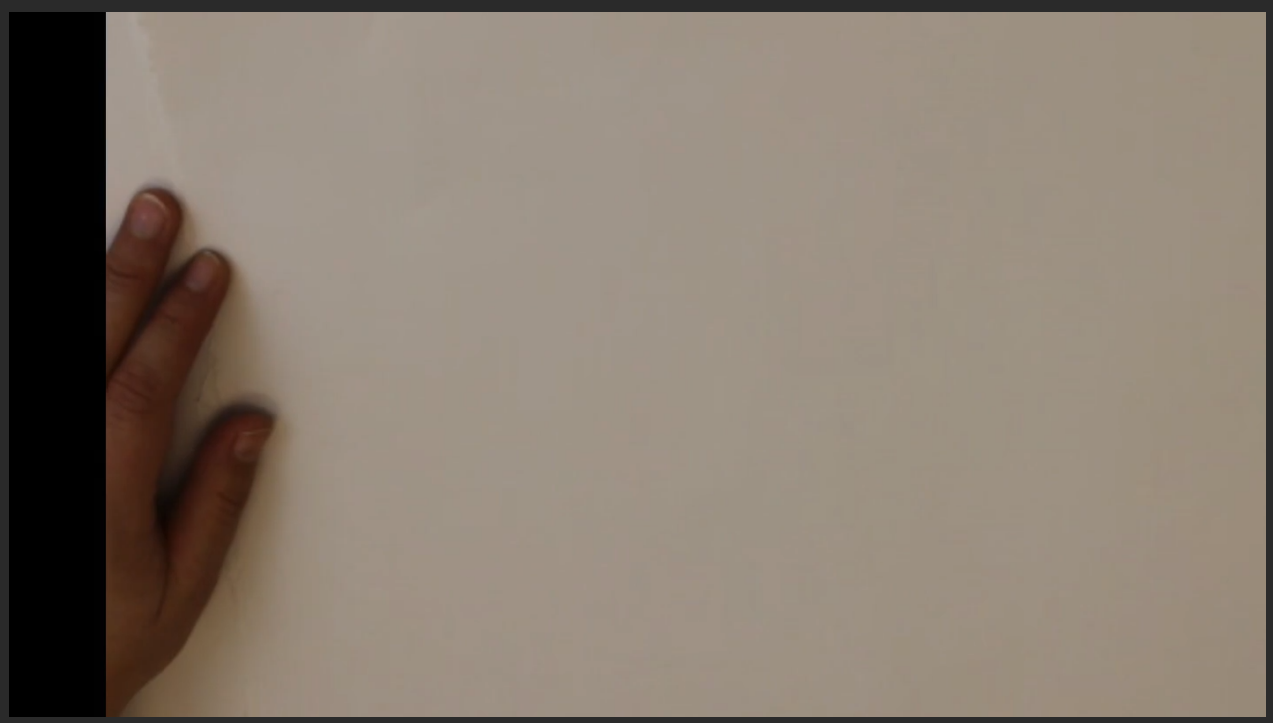
Media Queries



@media

CSS uses the @media rule to detect information about the device being used to access the website. For example, it can be used to detect the size of the device, its capabilities, and its orientation. For this course, we will be using it to detect the device size, but keep in mind that it is capable of much more.

Different devices

We have three basic categories of devices: phones, tablets, and desktops. Of course there are more, but these are the big three. We will focus on using media queries to detect which category of device is being used and respond appropriately. To do this, we'll need to know how we define each of these categories.

Flexible units

Responsive web design is all about making our website fluid and proportional, so sometimes it doesn't make sense to make elements on the page have a fixed size in pixels. We'll still use pixels - for example, we usually use pixels for padding, since we typically want the same amount of padding regardless of screen size - but now we will also use flexible units for the elements on the page that need to be fluid. For example, we can use **percentage (%)** to set an element to take a certain amount of the available width, and the available width is determined by screen size! Other flexible units include **em** and **rem**. The **rem** unit is a multiplier of the default font size of the user's browser. The **em** unit is a multiplier of the element's parent. Experiment with these different units to understand how they behave.

Example

Let's say that we have four buttons on our website that we would like to change color depending on the size of the screen. We'll give desktop users red buttons, tablet users blue buttons, and phone users yellow buttons. We'll give the buttons the class of *.navbtn*.

To get you started, here's how we would set up a media query in our CSS to notice if the user's screen is 480 px wide or smaller. If this is true, then we will target all elements with the class *.navbtn* and set their background to be yellow.

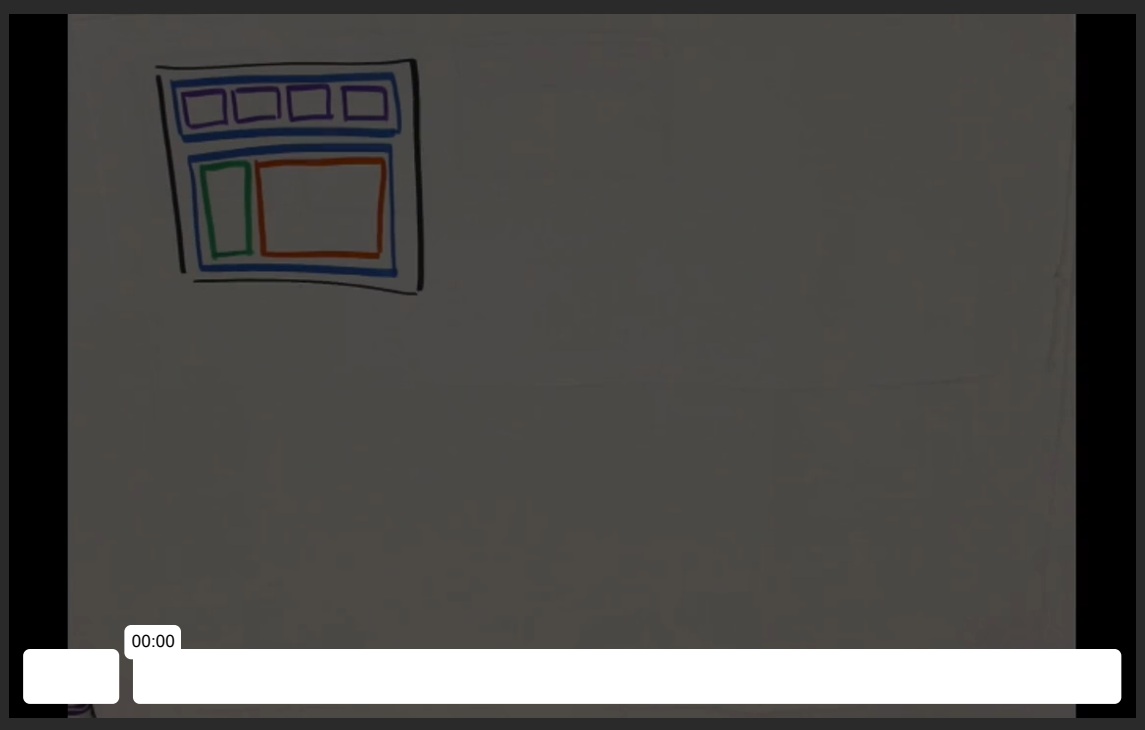
```
@media only screen and (max-width: 480px){
     /* we will set these stylings only if the device is a screen with a width of 480px or
less */
     .navbtn {
         background-color: yellow;
     }
}
```

Remember that CSS is read from top to bottom, so if the conditions of the media query are met, the stylings within could override any stylings you placed on *.navbtn* in the code above the media query. Any code that you write below the media query may override the stylings we did here.

Try changing the button colors from red to blue to yellow on your own, depending on screen size. Rewatch the video above if you need a hint. Then, change how the buttons are oriented depending on screen size.

- For desktop users, arrange all four buttons in one line
- For tablet users, place two buttons per line
- For phone users, give each button its own line

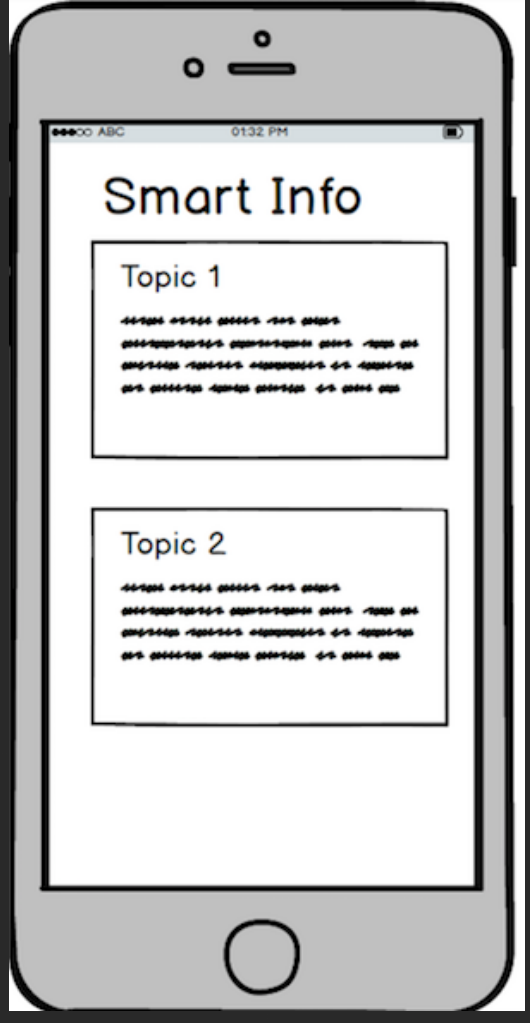
Grid System



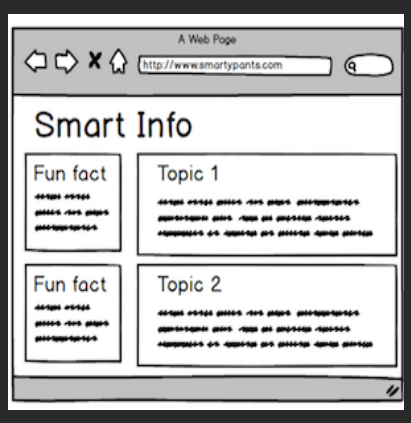
Why use the grid system?

To streamline the process of designing responsive websites, many developers choose to imagine that the screen is divided vertically into evenly distributed columns. Combined with dividing the screen into horizontal rows, this makes positioning our elements on the page much easier!

You don't need to use someone else's library just to have access to the grid system. You know how to use media queries, now just combine them with flexible units (for example, use percentages instead of pixels to set width) and you can make your own grid.



Here, we've thought about our website design with mobile first. We would want our data to be displayed very simply; just a vertical list that a user can easily scroll through on a phone. Using the grid idea, each topic gets all the available columns. For this example, assume that we are dividing our screen into four columns. Each column, therefore, gets 25% of the available width.



Now, we can think about a larger device. If we have more space, we can add a fun fact to each row. We'll give one column of the available four to this extra content and leave the remaining three for the main content.

This means that in our html, we'll give our elements different classes. For example, our optional elements should have a class that will make it possible for us to hide them when they are not wanted. To make our code easy to read, let's name our classes with a system so that we can identify for which screen size they are intended and how many columns they will occupy. Our classes will follow the pattern **screen size** - **number of columns**. For example, a class of *.med-1* will mean that for medium screens, the element will take up one column.

```
<div class="col sm-0 med-1">
   <h5>Did you know?</h5>
   <p>If you could fold a piece of paper 42 times, it would be tall enough to reach the
moon.</p>
</div>
```

In our css, we'll perform media queries that determine which classes we are using and which we are ignoring, depending on the screen size.

```
@media only screen and (max-width: 480px){
    .sm-0 {
        /* we will not display anything with the class .sm-0 if the screen's width is less
than 480px */
        display: none;
    }
```

Privacy Policy