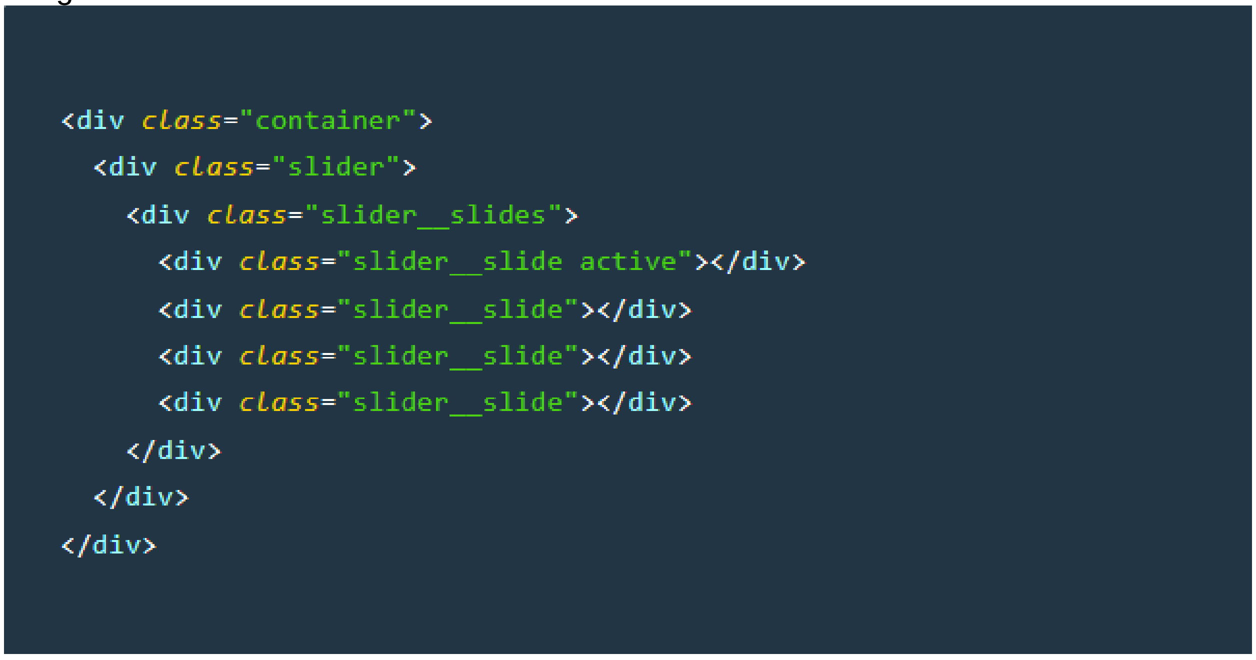
- How can a JavaScript image slider be created from scratch?
- How exactly do image slideshows using JavaScript operate?
- Are carousels really this year's JavaScript requirement?
- With the advancements in CSS, it is now possible to create a good slider using only CSS.

You still cannot accomplish much with CSS by itself, so you will need to use JavaScript to create your image slider if you want something more intricate.

The question that immediately arises is, "How do I build one?" Do not worry, after finishing this lesson, you will understand how to create a JavaScript image slider with the included source code!-, with breadcrumbs and navigation buttons included.

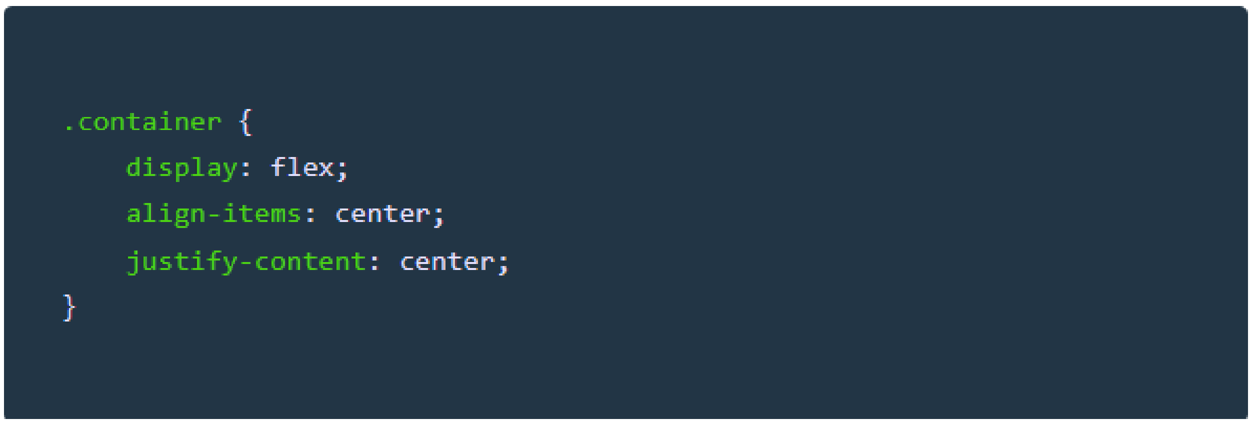
Create the Structure for the Image Slider with HTML

First, we will have to create the basic structure for the image slider. The html for our image slider will look like this:



```html
<div class="container">
  <div class="slider">
    <div class="slider__slides">
      <div class="slider__slide active"></div>
      <div class="slider__slide"></div>
      <div class="slider__slide"></div>
      <div class="slider__slide"></div>
    </div>
  </div>
</div>
```

Therefore, container is just the element that contains the future JavaScript images slideshow. This can be any part of your site, but I have just gone for a simple flexbox:



```css
.container {
    display: flex;
    align-items: center;
    justify-content: center;
}
```

Next is the main slider element. This will hold the image slides and the navigation elements:

```
.slider {
    display: block;
    position: relative;
    width: 100%;
    max-width: 900px;
    margin: 10px;
    background-color: white;
    overflow: hidden;
}
```

You can adjust the size in any way that floats your boat – just make sure there’s a position property applied to it.

We also have a container div for the slides, slider__slides and four slider__slide elements, which I’ve styled like this:

```
.slider__slides {
  width: 100%;
  padding-top: 66%;
}

.slider__slide {
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 50px;
    font-weight: bold;
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: lemonchiffon;
    transition: 1s;
    opacity: 0;
}

.slider__slide.active {
    opacity: 1;
}
```

Now, here is where it gets a little interesting. Setting padding-top to a percentage in slider__slides will maintain its aspect ratio.

The the slides themselves are positioned absolutely, top and left are both 0, while width and height are both 100%.

This means that each slide will completely fill the first parent element that has a position property applied to it (that is why I said make sure you do that to the slider element).

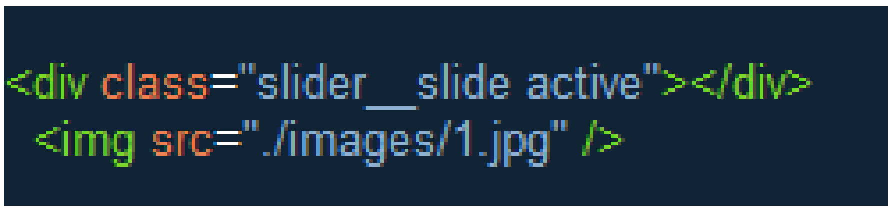
This also means that the four slides are stacked directly on top of each other. Notice that I have set the opacity of each slide to 0 – this makes them transparent. However, the .slider__slide.active part means that any element, which has both the slider__slide and active classes, will have an opacity of 1 – making the active slide visible.

Putting Images into the JavaScript Image Slideshows

There are a few ways to make an image slider in JavaScript.

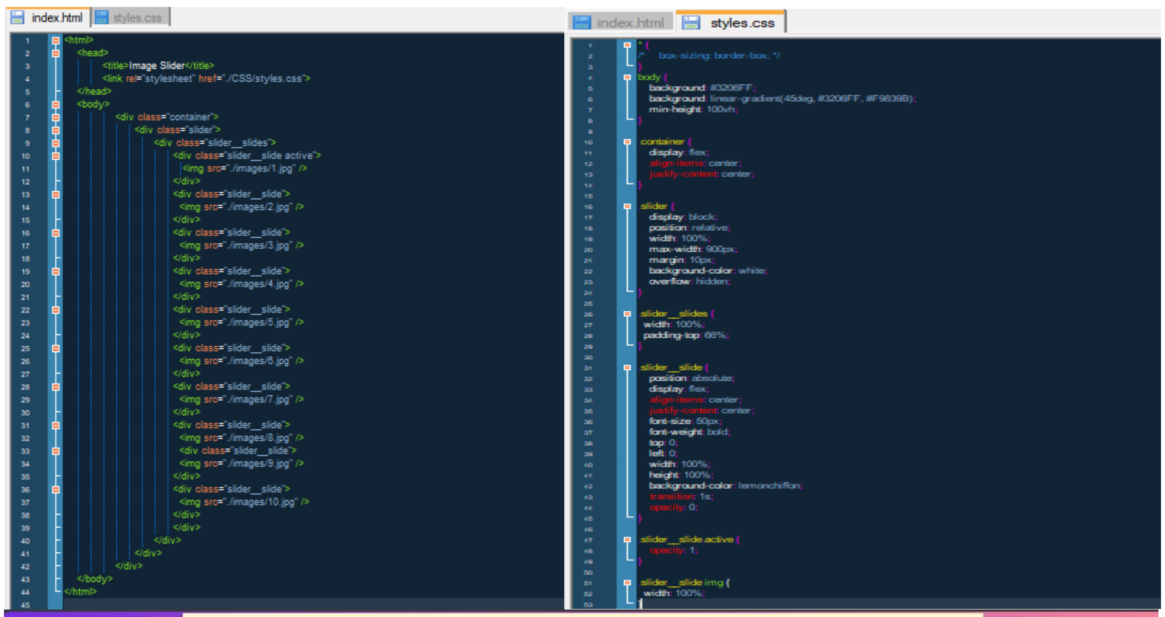
One is to add the images using the CSS background property, and have a separate class for each image. Another way is to have just have one slide, and change its background image directly with JS.

However, the approach we will take here is to use an img element inside each slide, like this:



```html
<div class="slider__slide active"></div>
  <img src="./images/1.jpg" />
```

I have added one image for each slide (compose of 10 images), but we can still only see the first image, the one with the active class:

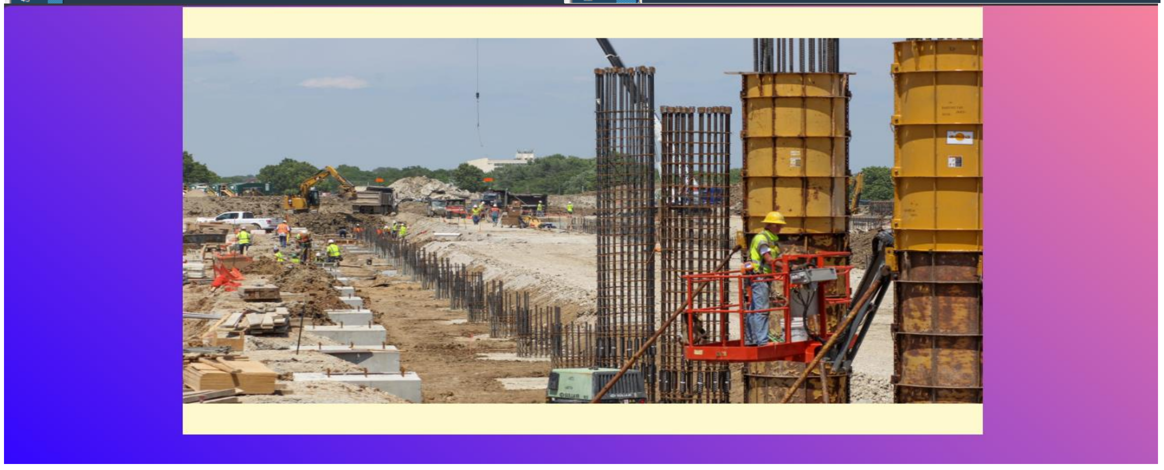


index.html

```html
<html>
  <head>
      <title>Image Slider</title>
      <link rel="stylesheet" href="./CSS/styles.css">
  </head>
  <body>
        <div class="container">
          <div class="slider">
            <div class="slider__slides">
              <div class="slider__slide active">
               <img src="./images/1.jpg" />
              </div>
              <div class="slider__slide">
               <img src="./images/2.jpg" />
              </div>
              <div class="slider__slide">
               <img src="./images/3.jpg" />
              </div>
              <div class="slider__slide">
               <img src="./images/4.jpg" />
              </div>
              <div class="slider__slide">
               <img src="./images/5.jpg" />
              </div>
              <div class="slider__slide">
               <img src="./images/6.jpg" />
              </div>
              <div class="slider__slide">
               <img src="./images/7.jpg" />
              </div>
              <div class="slider__slide">
               <img src="./images/8.jpg" />
               <div class="slider__slide">
               <img src="./images/9.jpg" />
              </div>
              <div class="slider__slide">
               <img src="./images/10.jpg" />
              </div>
              </div>
            </div>
          </div>
        </div>
  </body>
</html>
```

styles.css

```css
* {
/*    box-sizing: border-box; */
}
body {
  background: #3206FF;
  background: linear-gradient(45deg, #3206FF, #F9839B);
  min-height: 100vh;
}

.container {
  display: flex;
  align-items: center;
  justify-content: center;
}

.slider {
  display: block;
  position: relative;
  width: 100%;
  max-width: 900px;
  margin: 10px;
  background-color: white;
  overflow: hidden;
}

.slider__slides {
 width: 100%;
 padding-top: 66%;
}

.slider__slide {
  position: absolute;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 50px;
  font-weight: bold;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: lemonchiffon;
  transition: 1s;
  opacity: 0;
}

.slider__slide.active {
  opacity: 1;
}

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

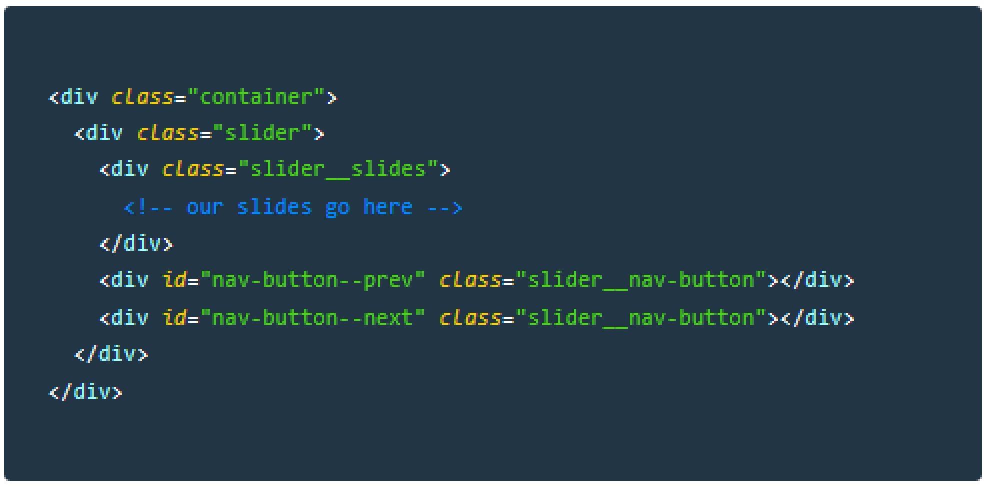


So now, let us add the navigation elements so people can cycle through the images on the slider.

Add Forwards and Backwards Buttons to the JavaScript Image Slider

Users look for navigation buttons at the far left and right edges of sliders – so that is where we will put them!

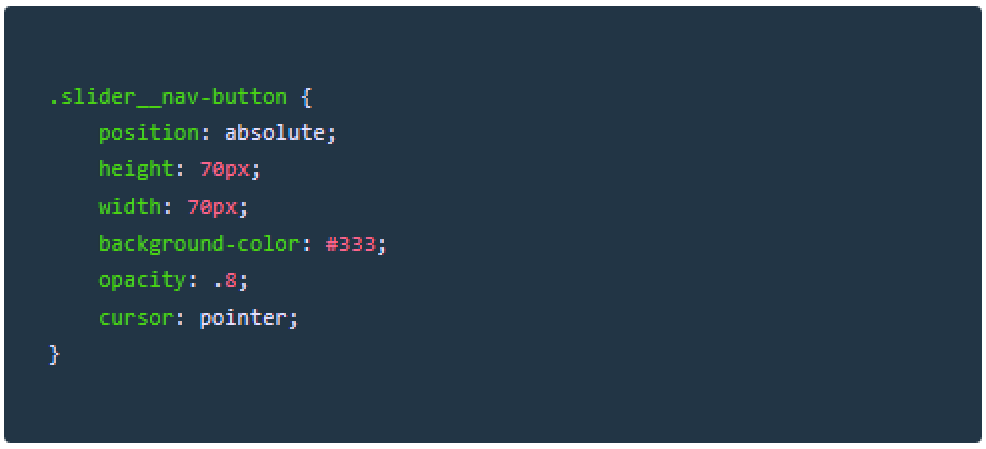
First, let us add these elements to the markup, inside slider, but below slider__slides:



```html
<div class="container">
  <div class="slider">
    <div class="slider__slides">
      <!-- our slides go here -->
    </div>
    <div id="nav-button--prev" class="slider__nav-button"></div>
    <div id="nav-button--next" class="slider__nav-button"></div>
  </div>
</div>
```

As well as the slider__nav-button class, each button has a unique id – as we are making our image slider using JavaScript, we will need these IDs so that we can select these elements within our JS code.

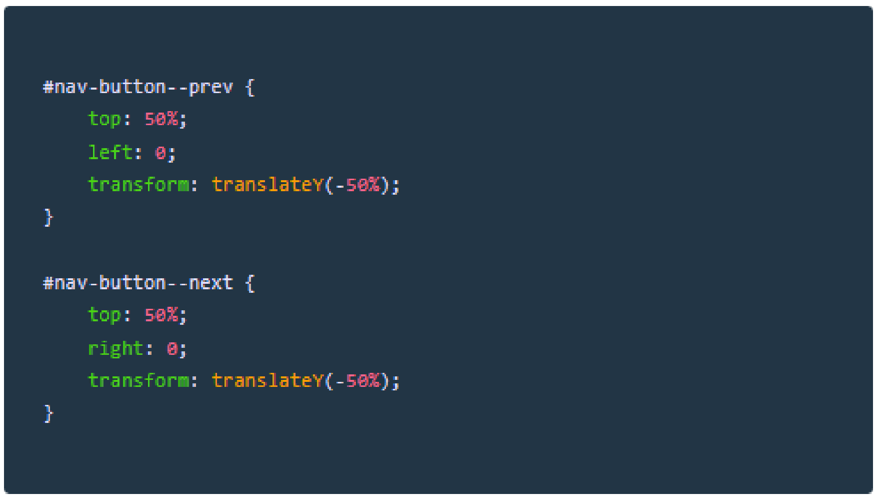
You can style your buttons however, you want, of course. Personally, I am just going to keep it simple with a grey box. Hey – sometimes the simple option is the best:



```css
.slider__nav-button {
    position: absolute;
    height: 70px;
    width: 70px;
    background-color: #333;
    opacity: .8;
    cursor: pointer;
}
```

I have set opacity to .8, so the buttons will be a little transparent and will not fully block whatever is behind them. In addition, since we want these to be clickable we need cursor: pointer so that the user knows they can click it.

Now, because these two buttons are at different sides of the slider, we will need to use slightly different CSS to position each one. We can use the ids for that:



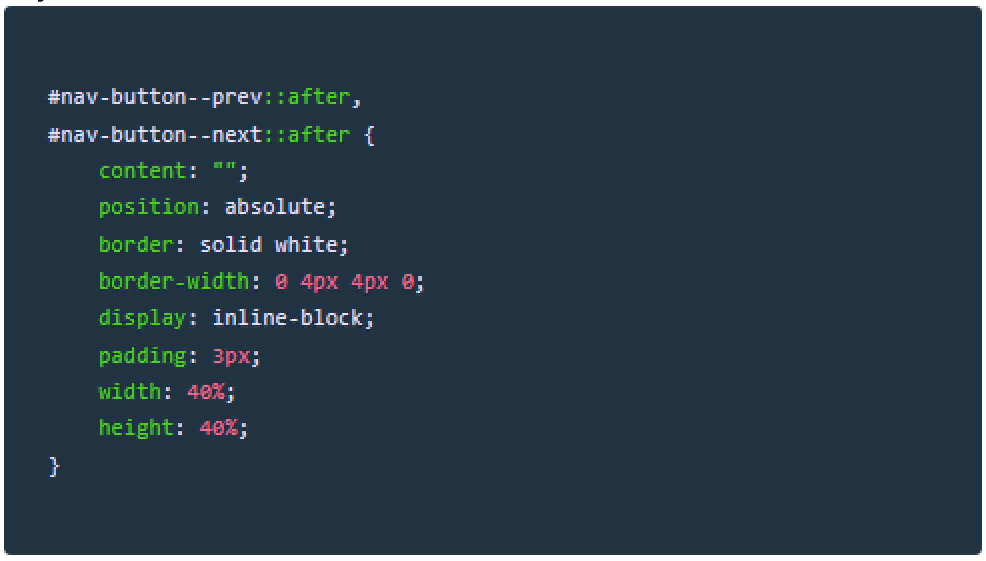
```css
#nav-button--prev {
    top: 50%;
    left: 0;
    transform: translateY(-50%);
}

#nav-button--next {
    top: 50%;
    right: 0;
    transform: translateY(-50%);
}
```

Setting top to 50% will put the top of the element halfway down the parent. Therefore, it will not be perfectly centered vertically – it will be a little lower than it should be.

The solution is to add transform: translate (-50%); When you translate by a percentage, you move the element by a proportion of its own size. Therefore -50% will perfectly center the button, not matter that its height happens to be.

Adding Arrows to the Navigation Buttons with CSS

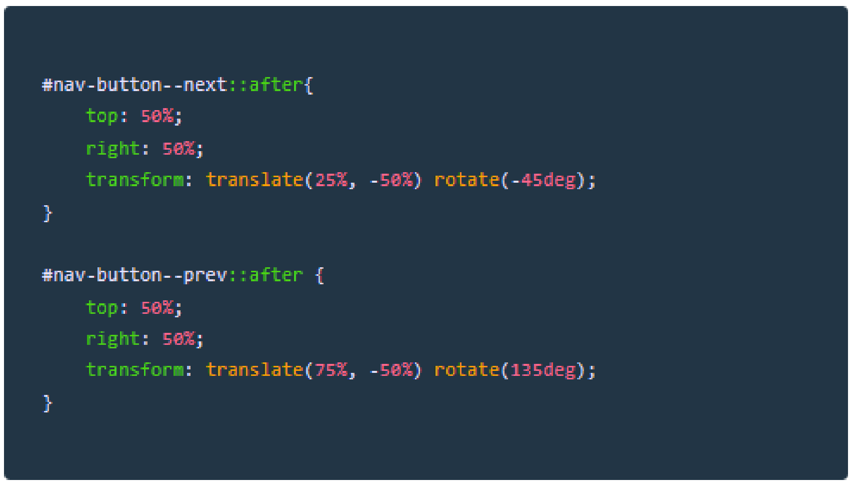
Now our navigation buttons are just grey boxes – we should indicate that these are buttons somehow. There are loads of ways of doing this – you could use an image of an arrow, an SVG, or even just write “Prev” and “Next”. However, here is another way:



```
#nav-button--prev::after,
#nav-button--next::after {
    content: "";
    position: absolute;
    border: solid white;
    border-width: 0 4px 4px 0;
    display: inline-block;
    padding: 3px;
    width: 40%;
    height: 40%;
}
```

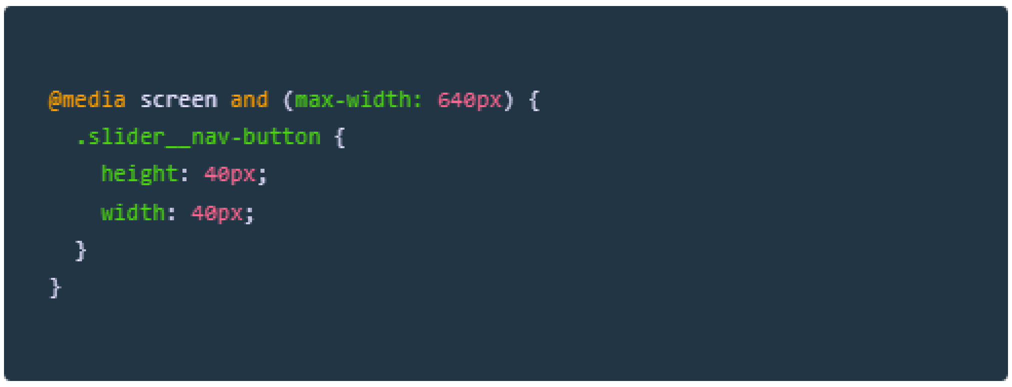
This uses the after pseudoelement, which means we can add it to the JavaScript images slideshow without having to add anymore html. It creates a box with a solid white border along only its right and bottom edges.

All we need to do now, is rotate the boxes so that the borders look like arrows, and use ‘translate’ to make sure they’re centred within the button:



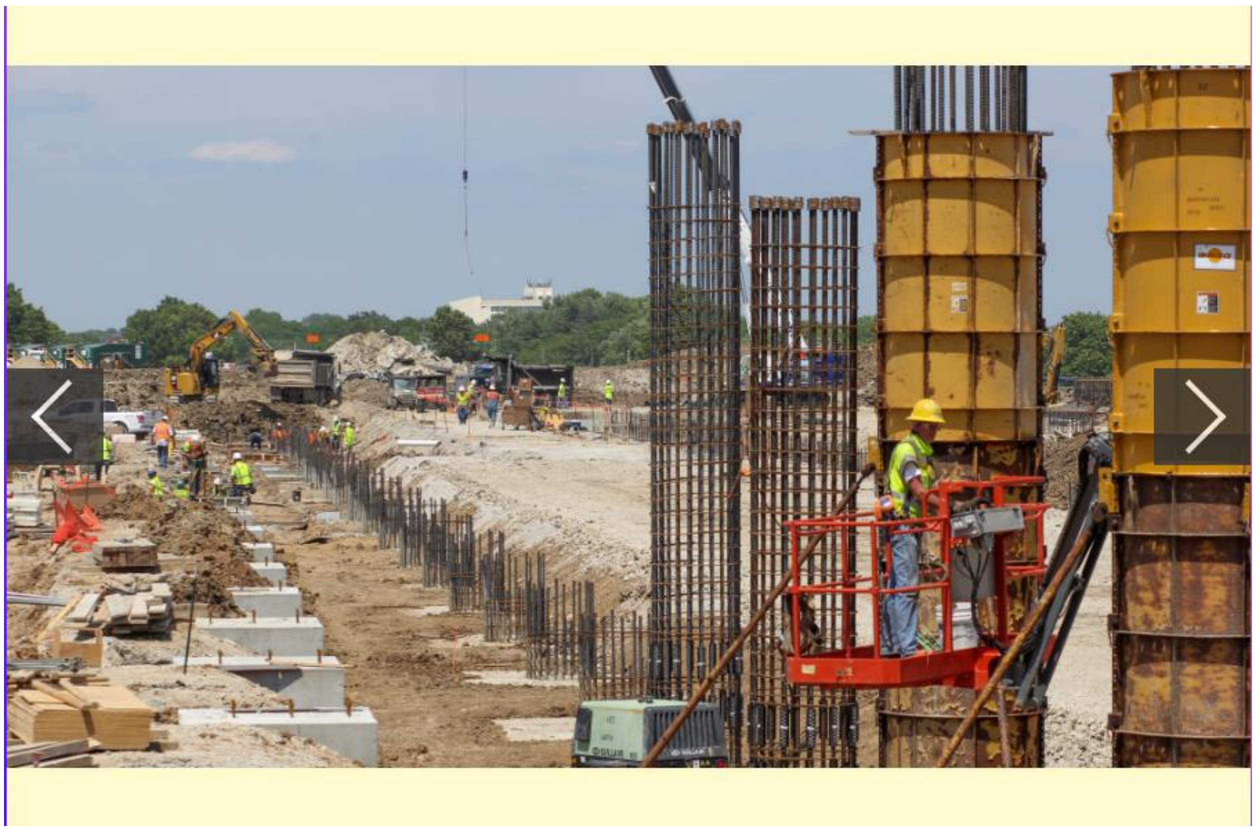
```
#nav-button--next::after{
    top: 50%;
    right: 50%;
    transform: translate(25%, -50%) rotate(-45deg);
}

#nav-button--prev::after {
    top: 50%;
    right: 50%;
    transform: translate(75%, -50%) rotate(135deg);
}
```

Moreover, let us add a media query to shrink these buttons a bit on smaller screens:



```
@media screen and (max-width: 640px) {
  .slider__nav-button {
    height: 40px;
    width: 40px;
  }
}
```

If we put all this together, her
e’s what we get:



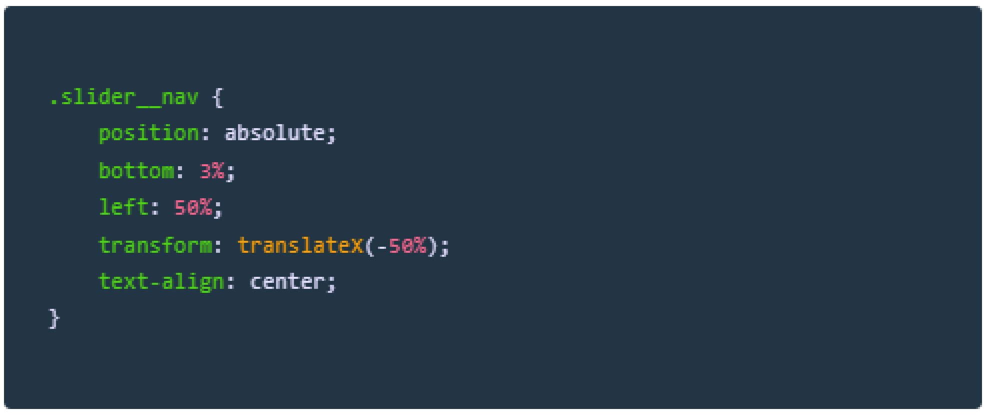
Add Breadcrumbs To The JavaScript Images Slideshow

Breadcrumbs (or navigation dots, bullets or whatever else you want to call them) is actually easy to add. First, let us update our HTML once again:

```
<div class="container">
  <div class="slider">
    <div class="slider__slides">
      <!-- our slides go here -->
    </div>
    <div id="nav-button--prev" class="slider__nav-button"></div>
    <div id="nav-button--next" class="slider__nav-button"></div>
    <div class="slider__nav">
      <div class="slider__navlink active"></div>
      <div class="slider__navlink"></div>
      <div class="slider__navlink"></div>
      <div class="slider__navlink"></div>
    </div>
  </div>
  </div>
</div>
```

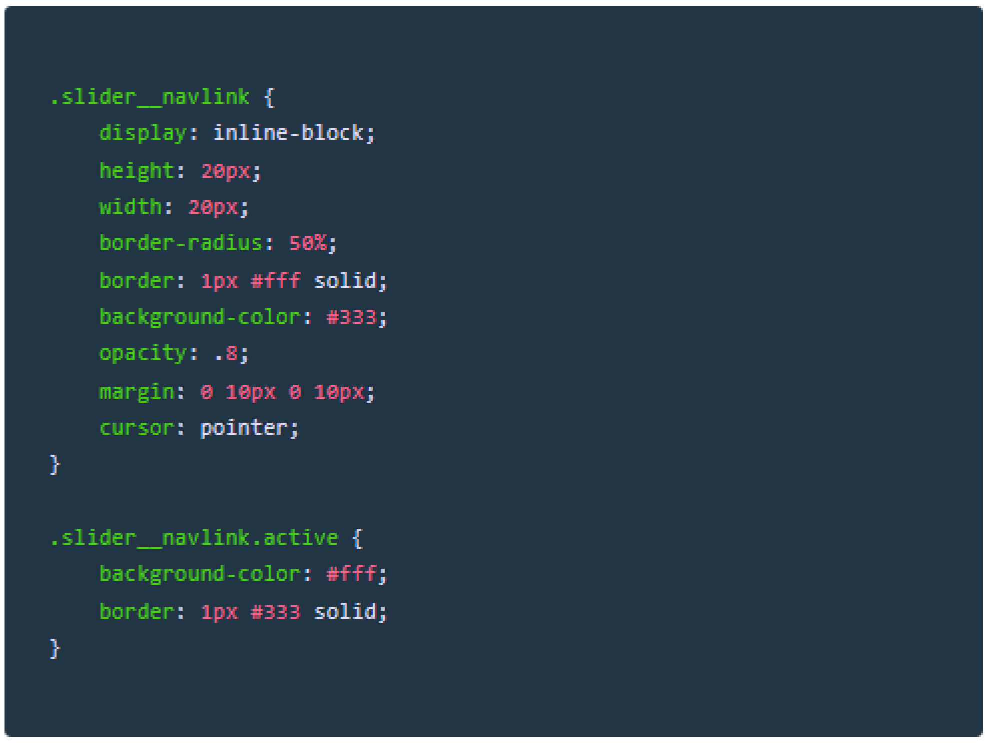
Now we have a slider__nav element, which contains four slider__navlink divs – one for each slide. The first one has an active class added to it also, which we can use to make the active breadcrumb stand out from the rest.

Traditionally, breadcrumbs like these go at the bottom of a slider:



```
.slider__nav {
    position: absolute;
    bottom: 3%;
    left: 50%;
    transform: translateX(-50%);
    text-align: center;
}
```

We have put the container for the breadcrumbs 3% away from the bottom of the image slider, and used the same centering trick we used with the buttons, only this time horizontally. Now for the breadcrumbs themselves:



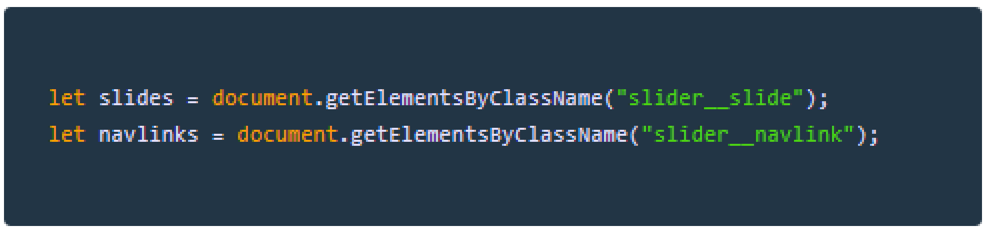
```
.slider__navlink {
    display: inline-block;
    height: 20px;
    width: 20px;
    border-radius: 50%;
    border: 1px #fff solid;
    background-color: #333;
    opacity: .8;
    margin: 0 10px 0 10px;
    cursor: pointer;
}

.slider__navlink.active {
    background-color: #fff;
    border: 1px #333 solid;
}
```

Simple stuff here – just using border-radius to make them circles, and setting the background color to white on the active slide. The rest are filled in with the same grey used in the buttons.

We are almost there – we just need to make these buttons actually do something! We will be using JavaScript for that.

Putting the JavaScript into the Image Slideshows

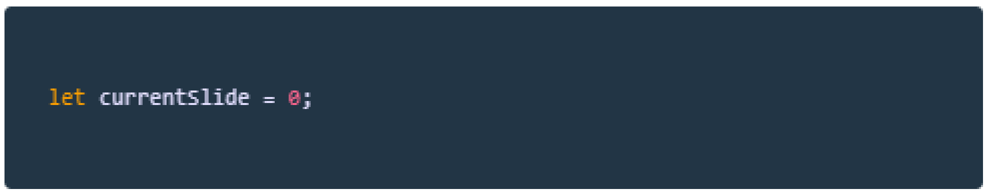
First, let’s make it easy to deal with our slides and breadcrumbs using JS:



```
let slides = document.getElementsByClassName("slider__slide");
let navlinks = document.getElementsByClassName("slider__navlink");
```

The first line goes through our markup, finds every element called slider__slide, and stores a pointer to it in an array called slides. The next line does the same for our breadcrumb elements, this time putting them in an array called navlinks.

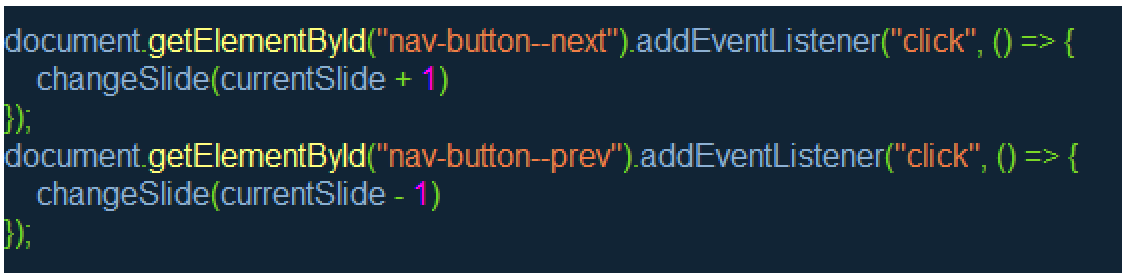
We also need to keep track of which slide is currently active, so lets make a variable for that:



```
let currentSlide = 0;
```

I’ve set this to 0 because as you may know, in JavaScript arrays are indexed from 0, not 1. So our first slide is in slides[0], the second is in slides[1], and so on.

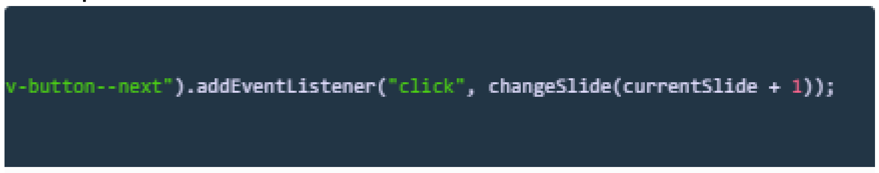
Now we need to add event listeners to each of our buttons:



```
document.getElementById("nav-button--next").addEventListener("click", () => {
  changeSlide(currentSlide + 1)
});
document.getElementById("nav-button--prev").addEventListener("click", () => {
  changeSlide(currentSlide - 1)
});
```

When a user clicks one of these buttons, our image slider JavaScript source code will call a function called changeSlide (which we’ll create in a minute), and pass an argument – currentSlide + 1 for the forwards button, and currentSlide - 1 for the backwards one. That lets us know which slide the user is trying to move to, so we can put the active class onto that slide, and take it off of the current slide.

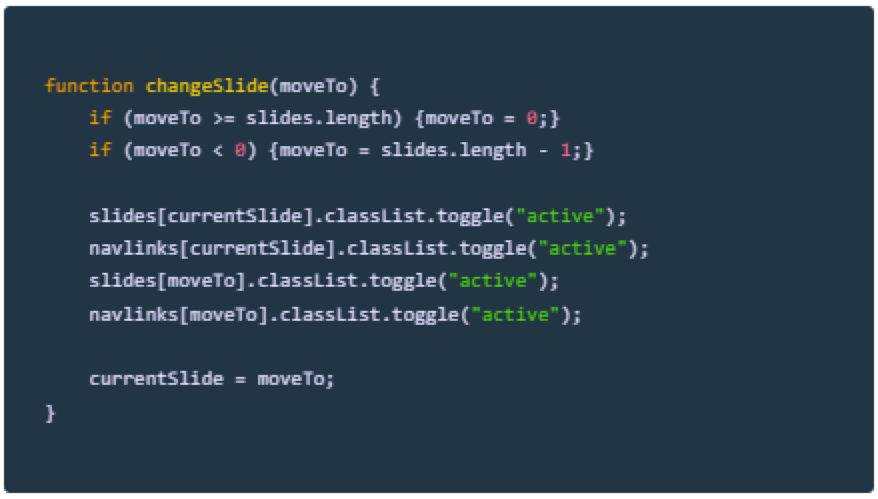
As you can see here, I’ve wrapped the function call within an anonymous function – this is necessary when you want to pass an argument using an event listener. For example, this doesn’t work:



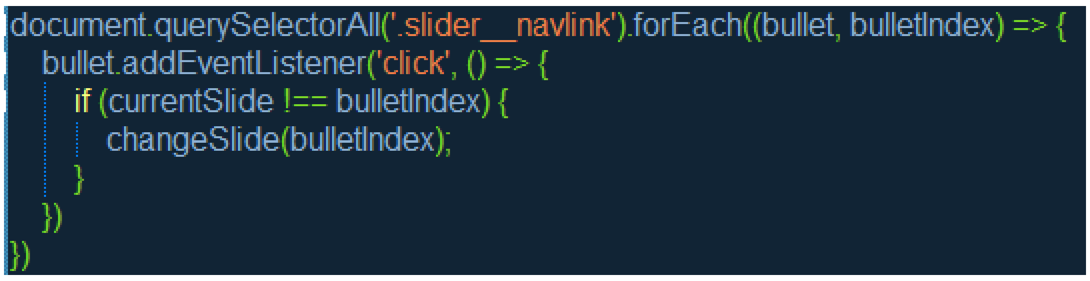
```
v-button--next").addEventListener("click", changeSlide(currentSlide + 1));
```

So that is why you have to put the function call within the () => { } block.

Creating the JavaScript Function That Will Change the Slide



```
function changeSlide(moveTo) {
    if (moveTo >= slides.length) {moveTo = 0;}
    if (moveTo < 0) {moveTo = slides.length - 1;}

    slides[currentSlide].classList.toggle("active");
    navlinks[currentSlide].classList.toggle("active");
    slides[moveTo].classList.toggle("active");
    navlinks[moveTo].classList.toggle("active");

    currentSlide = moveTo;
}
```


```
document.querySelectorAll('.slider__navlink').forEach((bullet, bulletIndex) => {
  bullet.addEventListener('click', () => {
    if (currentSlide !== bulletIndex) {
      changeSlide(bulletIndex);
    }
  })
})
```

Here we've used querySelectorAll to grab all of the slider__navlink elements, and then we're using forEach to loop through them.

So on the first loop, bullet will point to the first slide, and bulletIndex will equal 0.
On the next loop, bullet will point to the second slide, and bulletIndex will equals 1.
And so on.

The idea is, if they click a button, we know they want to change to slide bulletIndex – so we can just pass that value to our funtion with changeSlide(bulletIndex).

However, we'd better first check if they are already on slide bulletIndex. There's no point fading out and then back in to the same slide. So the if statement ensures we'll only call changeSlide, if currentSlide does not equal bulletIndex.

Final Output: