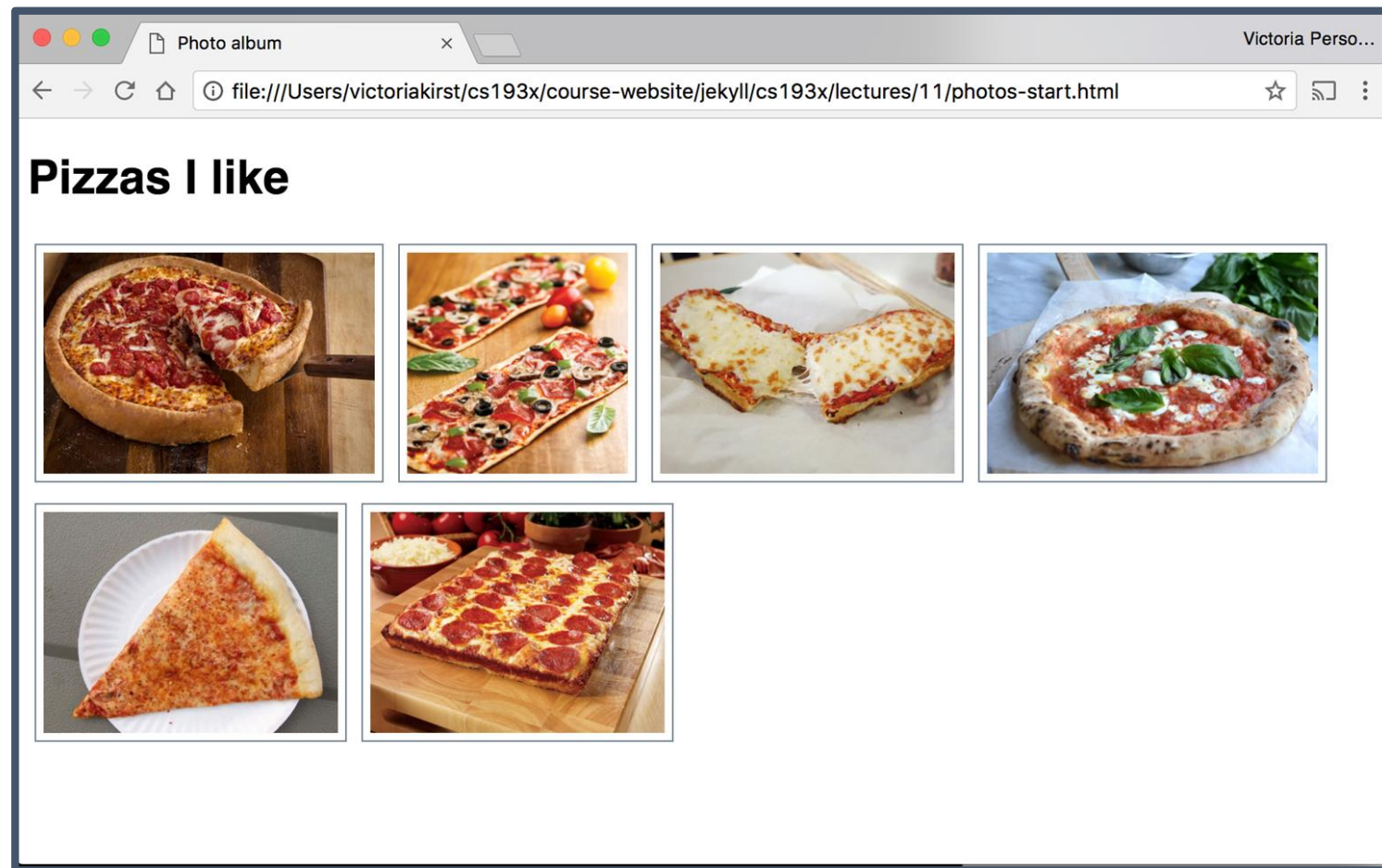


# Case study: Photo album

Keyboard events

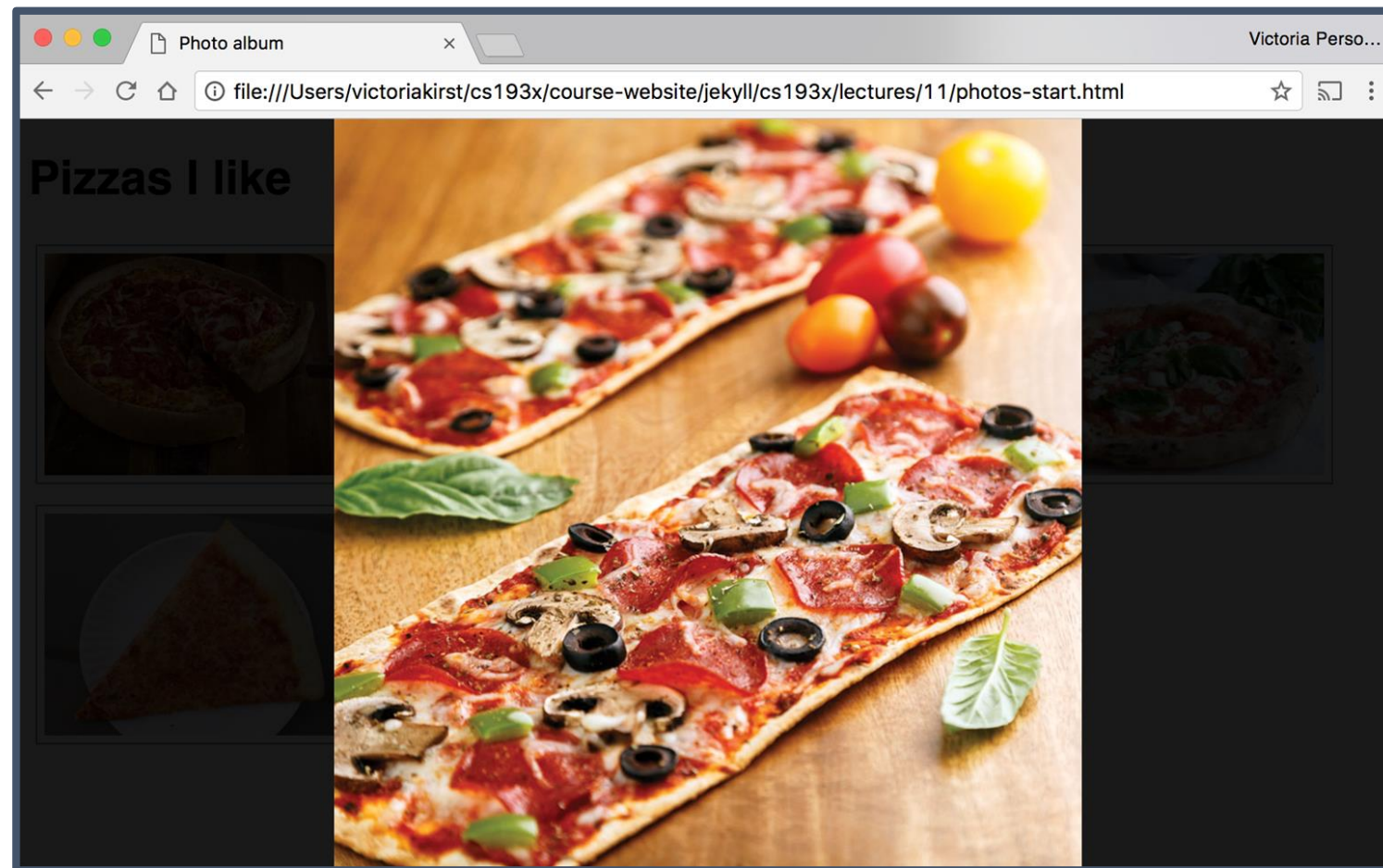
# Example: Photo Album

We're going to add a few features to this photo album:



# Example: Photo Album

We're going to add a few features to this photo album:



Code walkthrough:

[photo-start.html](#)

[photo.js](#)

[photo.css](#)

# General setup

```
<body>
  <h1>Pizzas I like</h1>
  <section id="album-view">
  </section>

  <section id="modal-view" class="hidden">
  </section>
</body>
```

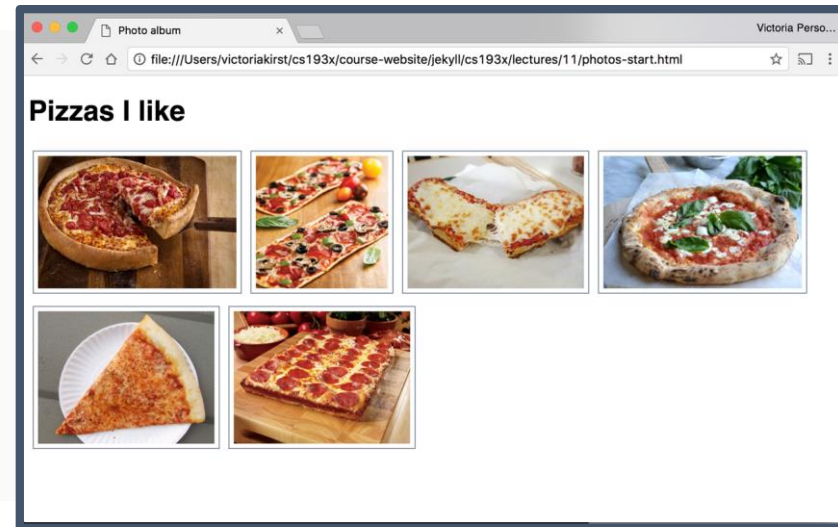
[photo.html](#) contains both "screens":

- The album view: Thumbnails of every photo
- The "[modal](#)" view: A single photo against a semi-transparent black background
  - Hidden by default

# CSS: Album

[photo.css](#): The album view CSS is pretty straightforward:

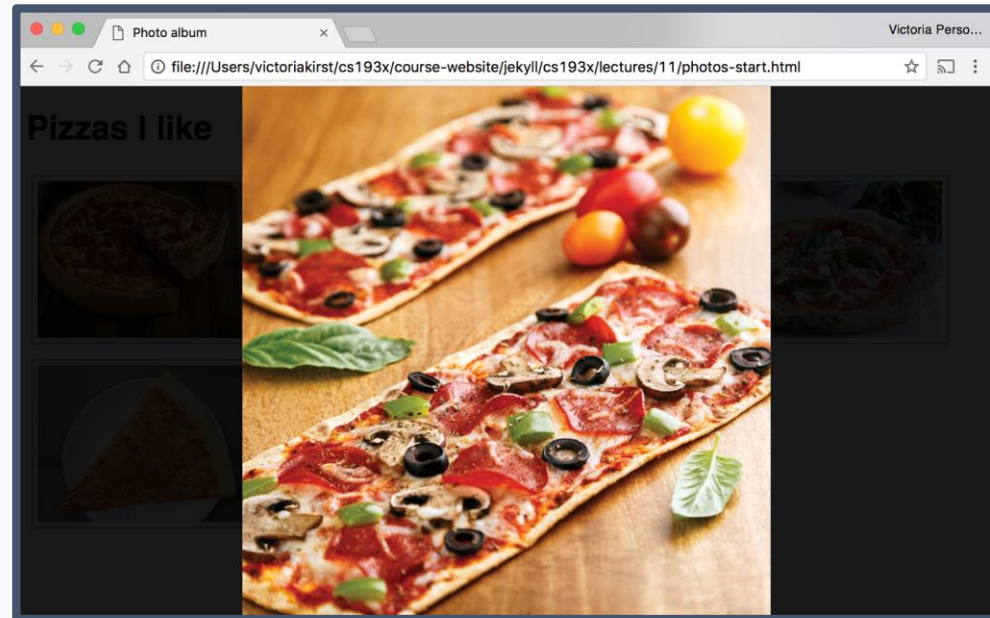
```
#album-view img {  
  border: 1px solid slategray;  
  margin: 5px;  
  padding: 5px;  
  height: 150px;  
}
```



# CSS: Modal

Modal view is a little more involved, but all stuff we've learned:

```
#modal-view {  
  position: absolute;  
  top: 0;  
  left: 0;  
  height: 100vh;  
  width: 100vw;  
  
  background-color: rgba(0, 0, 0, 0.9);  
  z-index: 2;  
  
  display: flex;  
  justify-content: center;  
  align-items: center;  
}
```





# CSS: Modal image

```
#modal-view img {  
  max-height: 100%;  
  max-width: 100%;  
}
```

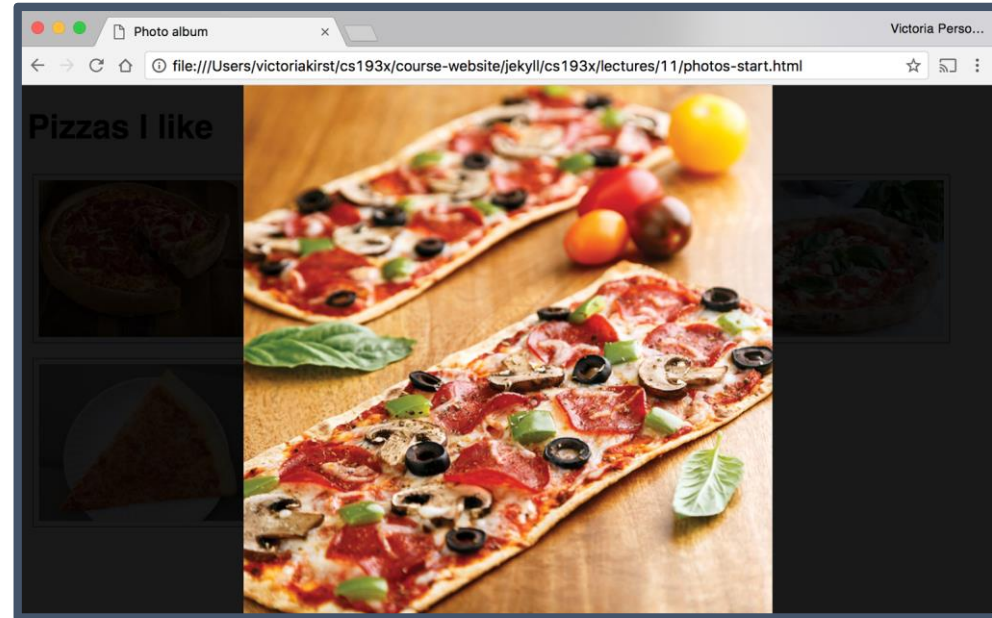


Image sizes are constrained to the height and width of the parent, `#modal-view` (whose height and width are set to the size of the viewport)

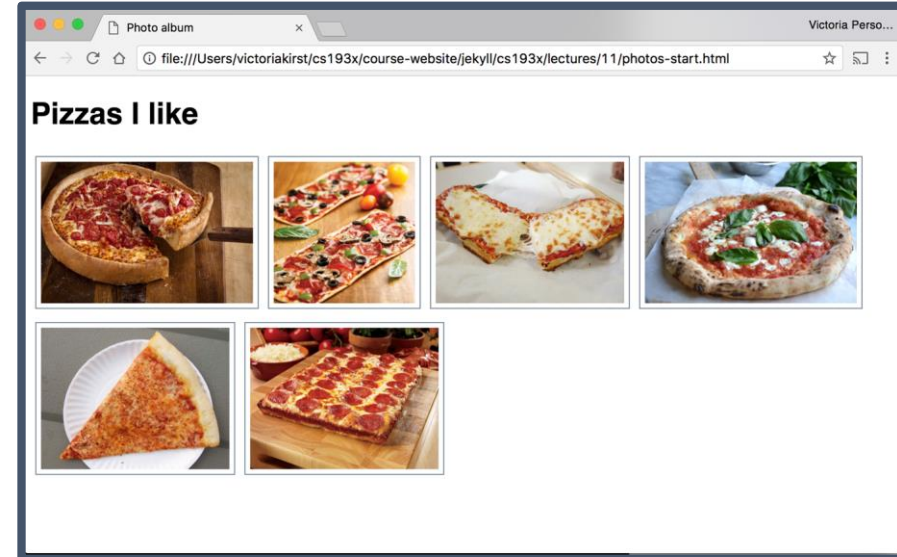


# CSS: Hidden modal

```
<body>
  <h1>Pizzas I like</h1>
  <section id="album-view">
  </section>

  <section id="modal-view" class="hidden">
  </section>
</body>
```

```
#modal-view.hidden {
  display: none;
}
```



Even though both the album view and modal view are in the HTML, the modal view is set to `display: none;` so it does not show up.

# Global List of Photos

```
<head>
  <meta charset="utf-8">
  <title>Photo album</title>
  <link rel="stylesheet" href="css/photo.css">
  <script src="js/photo-list.js" defer></script>
  <script src="js/photo.js" defer></script>
</head>
```

```
const PHOTO_LIST = [
  'images/deepdish.jpg',
  'images/flatbread.jpg',
  'images/frenchbread.jpg',
  'images/neapolitan.jpg',
  'images/nypizza.jpg',
  'images/squarepan.jpg'
];
```

[photo-list.js](#): There is a global array with the list of string photo sources called PHOTO\_LIST.

# Photo thumbnails

```
function createImage(src) {  
  const image = document.createElement('img');  
  image.src = src;  
  return image;  
}
```

```
const albumView = document.querySelector('#album-view');  
for (let i = 0; i < PHOTO_LIST.length; i++) {  
  const photoSrc = PHOTO_LIST[i];  
  const image = createImage(photoSrc);  
  image.addEventListener('click', onThumbnailClick);  
  albumView.appendChild(image);  
}
```

[photo.js](#): We populate the initial album view by looping over PHOTO\_LIST and appending <img>s to the #album-view.

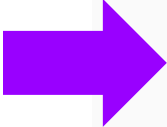
# Clicking a photo

```
function onThumbnailClick(event) {  
  const image = createImage(event.currentTarget.src);  
  modalView.appendChild(image);  
  modalView.classList.remove('hidden');  
}
```

When the user clicks a thumbnail:

- We create another `<img>` tag with the same src
- We append this new `<img>` to the `#modal-view`
- We unhide the `#modal-view`

# Positioning the modal



```
function onThumbnailClick(event) {  
  const image = createImage(event.currentTarget.src);  
  modalView.style.top = window.pageYOffset + 'px';  
  modalView.appendChild(image);  
  modalView.classList.remove('hidden');  
}
```

We'll add another line of JavaScript to anchor our modal dialog to the top of the viewport, not the top of the screen:

```
modalView.style.top = window.pageYOffset + 'px';
```

(See [window.pageYOffset mdn](#). It is the same as [window.scrollY](#).)

# Aside: style attribute

Every [HTMLElement](#) has a [style](#) attribute that lets you set a style directly on the element:

```
element.style.top = window.pageYOffset +  
'px';
```

Generally **you should not use the style property**, as adding and removing classes via [classList](#) is a better way to change the style of an element via JavaScript

But when we are setting a CSS property based on JavaScript values, we must set the `style` attribute directly.

# No scroll on page



```
function onThumbnailClick(event) {  
  const image = createImage(event.currentTarget.src);  
  document.body.classList.add('no-scroll');  
  modalView.style.top = window.pageYOffset + 'px';  
  modalView.appendChild(image);  
  modalView.classList.remove('hidden');  
}
```

```
.no-scroll {  
  overflow: hidden;  
}
```

And we'll also set `body { overflow: hidden; }` as a way to disable scroll on the page.



# Closing the modal dialog

```
function onModalClick() {  
    document.body.classList.remove('no-scroll');  
    modalView.classList.add('hidden');  
    modalView.innerHTML = '';  
}
```

```
const modalView = document.querySelector('#modal-view');  
modalView.addEventListener('click', onModalClick);
```

When the user clicks the modal view:

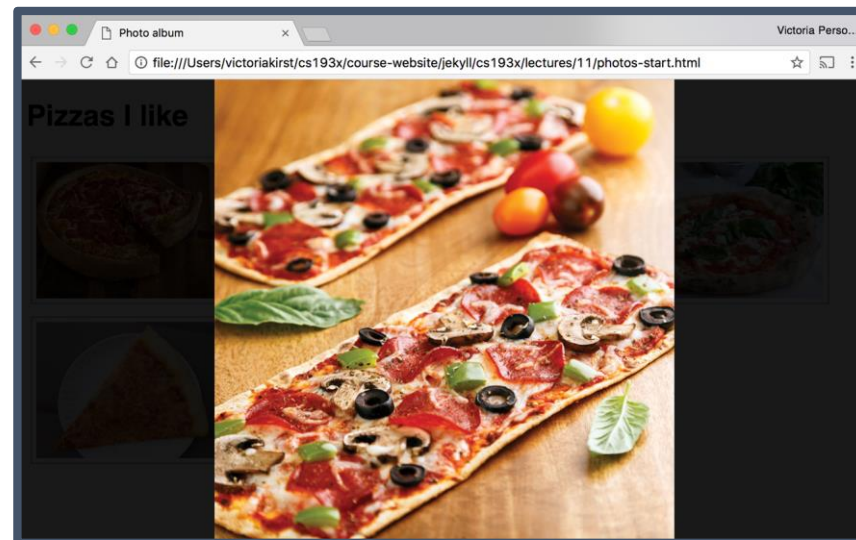
- We hide the modal view again
- We enable scroll on the page again
- We clear the image we appended to it by setting `innerHTML = ''`;

Adding keyboard navigation

# Navigating photos

Let's add some keyboard events to navigate between photos in the Modal View:

- Left arrow: Show the "i - 1"th picture
- Right arrow: Show the "i + 1"th picture
- Escape key: Close dialog



Finished result:  
[photo-mobile-finished.html](#)

# Object-oriented photo album

Let's look at an object-oriented version of the photo album:

[CodePen](#) / [Debug](#)

