How to Program a Photo Gallery
by Carl J. Roberts

Who this guide is for:

1. Developers who understand HTML, CSS, JavaScript, and jQuery.
2. Developers who want to improve their programming skills by creating a photo gallery.

Who this guide is *not* for:

1. Developers who don't understand HTML, CSS, JavaScript, and jQuery.
2. Cats, because they have no thumbs, and typing is difficult.

What You'll Use:

- HTML
- JavaScript
- CSS
- jQuery
- FontAwesome Icons
- Your favorite photos.

Sections:

1. Setup
2. HTML Markup: gallery.html
3. CSS Styles: gallery.css
4. JavaScript: gallery.js

Section 1: Setup

1. Create a folder on your computer named *photogallery* where we'll store our code.
2. Using your favorite text editor or IDE, create the following within the *photogallery* folder
    a. gallery.html
    b. gallery.js
    c. gallery.css

3. Copy photos to the *photogallery* folder.

Section 2: HTML Markup: *gallery.html*

Let's get the boring HTML markup out of the way. Open *gallery.html* in your browser, and also in your IDE / text editor. As with any guide, it would be ideal to go through each part in detail, and HTML markup is no exception.

Contents of *gallery.html*

```
<!doctype html>
<html>
 <head>
   <title>Photo Gallery</title>
   <meta charset=”UTF-8”/>
   <meta name=”viewport” content=”width=device-width, initial-scale=1”/>
   <link rel="stylesheet"
 href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"/>
   <link rel=”stylesheet” href=”gallery.css”/>
   <script src="https://code.jquery.com/jquery-3.3.1.min.js”></script>
     <script src=”gallery.js”></script>
   </head>
 <body>
   <div class=”page”>
       <h1>Photo Gallery</h1>
       <div class=”gallery”>
         <figure>
         <div class=”gallery-icon”>
           <a href=”whale1.jpg”>
             <img src=”whale1.jpg”/>
           </a>
         </div>
       </figure>
       <figure>
         <div class=”gallery-icon”>
           <a href=”whale2.jpg”>
             <img src=”whale1.jpg”/>
           </a>
         </div>
       </figure>
     </div>
   </div>
</body>
</html>
```

When we look at code, we'll go over the most important parts. Remember, this guide assumes you know HTML, JavaScript, and CSS.

#6: Responsive Scaling

```
<meta name=”viewport” content=”width=device-width, initial-scale=1”/>
```

This markup instructs devices to scale the *document's width* to the *perceived width* of your device. For example, your device may have a *resolution* of 1920x1080, but the *perceived width* is 360 pixels. In other words, your document's width then becomes 360 pixels.

#7: Include FontAwesome

```
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"/>
```

We include the FontAwesome stylesheet to style our gallery buttons.

#8: Gallery Stylesheet, gallery.css

```
<link rel=”stylesheet” href=”gallery.css”/>
```

We include our own stylesheet, to design how the gallery will look.

#9: jQuery

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js”></script>
```

This adds jQuery to our page. Place this *before* the reference to gallery.js, as noted in the code.

Tip: If using jQuery in a SaaS environment, it's ideal to save and reference a copy within your website. If you don't need 100% guarantee that it's available, you can take advantage of the CDN (as we have); users may already have this file cached in their device/computer, so it will load faster.

#10: JavaScript: gallery.js

```
<script src=”gallery.js”></script>
```

Include the gallery JavaScript file. Place this *after* the reference to jQuery.

#13 - #31: Gallery HTML
The markup here holds thumbnails of our photo gallery.

Tip: For each image, and faster loading on devices, create thumbnails of your images for the *src* of the *img* element. Keep the anchors pointing to the full-size image. For example:

```
<a href=”whale1.jpg”>
  <img src=”whale1-thumbnail.jpg”/>
</a>
```

Section 3: CSS Styles – *gallery.css*

We're going to make the gallery responsive, both to window sizes and mobile devices. There are 100+ lines to review.

Contents of *gallery.css*

```
1. * {
2.  box-sizing: border-box;
3. }
4.
5. html, body {
6.  margin: 0;
7.  padding: 0;
8.  min-width: 100%;
9.  min-height: 100%;
10. }
11.
12. h1 {
13.    text-align: center;
14.    color: #444444;
15. }
16.
17. .gallery {
18.    display: flex;
19.    flex-flow: row wrap;
20.    align-items: center;
21.    justify-content: center;
22. }
23.
24. .gallery > figure {
25.    margin: 0;
26.    padding: 0;
27.    flex: 0;
28. }
29.
30. .gallery > figure > .gallery-icon {
31.    border: solid 1px #999999;
32.    width: 150px;
33.    height: 150px;
34. }
35.
36. .gallery > figure > .gallery-icon > a {
37.    display: block;
38.    width: 100%;
39.    height: 100%;
40. }
41.
42. .gallery > figure > .gallery-icon > a > img {
43.    width: auto;
44.    height: auto;
45.    max-width: 100%;
46.    max-height: 100%;
47.    transition: opacity 150ms linear;
48. }
49.
50. .gallery > figure > .gallery-icon > a:hover > img {
51.    opacity: 0.9;
52. }
53.
54. .modal-holder {
```

```
55.    position: fixed;
56.    top: 0;
57.    left: 0;
58.    right: 0;
59.    bottom: 0;
60.    margin: auto;
61.    width: 100%;
62.    height: 100%;
63.    z-index: 10;
64.    background-color: rgba(0,0,0,0.95);
65.    transform: scale(0,0);
66.    transition: transform 150ms linear;
67.    overflow: hidden;
68. }
69.
70. .modal-holder.shown {
71.    transform: scale(1,1);
72. }
73.
74. .modal-holder > .modal-content {
75.    position: absolute;
76.    width: calc(100% - 20px);
77.    height: calc(100% - 20px);
78.    top: 0;
79.    left: 0;
80.    right: 0;
81.    bottom: 0;
82.    margin: auto;
83.    z-index: 1;
84. }
85.
86. .modal-holder > .modal-content > img {
87.    position: absolute;
88.    top: 0;
89.    left: 0;
90.    right: 0;
91.    bottom: 0;
92.    margin: auto;
93.    max-width: 100%;
94.    max-height: 100%;
95.    width: auto;
96.    height: auto;
97. }
98.
99. .modal-holder > .modal-controls {
100.   position: absolute;
101.   width: 100%;
102.   height: 50px;
103.   left: 0;
104.   right: 0;
105.   bottom: 0;
106.   z-index: 2;
107.   background-color: rgba(0,0,0,0.5);
108.   display: flex;
109.   flex-flow: row wrap;
110.}
111.
112..modal-holder > .modal-controls > a {
113.   flex: 1;
114.   font-size: 24px;
115.   color: #FFFFFF;
116.   text-align: center;
117.   line-height: 50px;
```

```
  height: 50px;
  text-decoration: none;
}

.modal-holder > a.modal-close {
  position: absolute;
  top: 5px;
  right: 5px;
  width: 50px;
  height: 50px;
  font-size: 24px;
  color: #FFFFFF;
  text-align: center;
  line-height: 50px;
  text-decoration: none;
  z-index: 3;
}
```

Important CSS lines explained

#1-3: Box-Sizing

```
* {
  box-sizing: border-box;
}
```

box-sizing: border-box; ensures that the dimensions of an element include its padding, border, and content. With the default, the following CSS would create a paragraph element with 100% width, plus overflowing the text on the left/right side by 5 pixels:

```
p {
  width: 100%;
  padding: 0px 5px;
}
```

#5-#9: Document Sizing

```
html, body {
  margin: 0;
  padding: 0;
  min-width: 100%;
  min-height: 100%;
}
```

Ensures the document is at least as wide and tall as the browser allows.

#17 - #22: Gallery Container

```
.gallery {
  display: flex;
  flex-flow: row wrap;
  align-items: center;
  justify-content: center;
}
```

The *.gallery* class flows each child element to the center of the gallery.
display: flex instructs browsers to make the *dimensions* of child elements to be flexible.

Tip: *flex-flow: row wrap* instructs browsers to flow the elements from *left-to-right*, along rows, or in a horizontal manner. *flex-flow: column wrap* instructs browsers to flow the elements from top-to-bottom.

```
flex-flow: row wrap;
```

flex-flow: row-reverse reverses the order of the child elements that are *within that row already*. What this means is it doesn't show the last photo in the gallery first, but, takes the *elements that sit within the row as it is seen, and reverses* ***just those ones***. See this example, where the photos are numbered 4-1.

```
flex-flow: row-reverse wrap;
```

#24 - #28: Figure

```
.gallery > figure {
   margin: 0;
   padding: 0;
   flex: 0;
}
```

We use *figure* elements to contain not only the thumbnail, but optional HTML, such as *captions*. Here, we're removing the default margin and padding of the *figure* element, and then instructing flex to *not resize* these elements – the children of the *figure* element will determine the size.

#30 - #34: Gallery Thumbnail Containers

```
.gallery > figure > .gallery-icon {
   border: solid 1px #999999;
   width: 150px;
   height: 150px;
}
```

We set the dimensions of our thumbnails to ensure they are all sized the same. Thumbnails of different sizes may look pretty for a collage, but for a gallery with dozens of photos, it can be annoying to try and locate a specific photo. Also, organized things can look more pleasing to most users.

#36 - #40: Thumbnail Anchor

```
.gallery > figure > .gallery-icon > a {
  display: block;
  width: 100%;
  height: 100%;
}
```

An anchor will be wrapped around each thumbnail *img* element. By using *display: block*, and *width/height: 100%*, we ensure the clickable area matches that of the thumbnail container. Since thumbnails, and photos, may have different height/width (150x75, for example), we want the same *clickable area* for the user, making it easier to select a photo to view.

#42 - #48: Thumbnail

```
.gallery > figure > .gallery-icon > a > img {
  width: auto;
  height: auto;
  max-width: 100%;
  max-height: 100%;
  transition: opacity 150ms linear;
}
```

Setting *img* elements to width/height: auto, max-width/max-height: 100%, ensures the image stays within the container (our anchor), and the height/width scales proportionately.

transition: opacity 150ms linear; is used to define the animating opacity of each thumbnail, which provides user feedback of which thumbnail the cursor is on, and which photo they will view if they click the mouse.

opacity implies that the only animation we want to see is applied to the *opacity* of the *img* element.
150ms refers to the total time taken for the animation to complete.
linear instructs the animation to have an equal speed/time for each frame the user sees.

#50 – 52: Thumbnail Highlighting

```
.gallery > figure > .gallery-icon > a:hover > img {
  opacity: 0.9;
}
```

We want the user to know what thumbnail is under their mouse cursor, and which photo they will see if they click the mouse. This is part of UI / UX design principles: providing the user with feedback.

We place the *:hover* on the *anchor* and not the *img* for a specific reason: The thumbnail may be smaller than 150x150, and since our *clickable* area is the entire anchor, which will always be 150x150, we want the user to feel that when the mouse cursor changes, the same effect happens, regardless of the actual size of the thumbnail. Otherwise, the user may have to move their cursor in a zig-zag way to know what photo they will view when they click the mouse.

#54 – 68: Photo Frame

```
.modal-holder {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
  width: 100%;
  height: 100%;
  z-index: 10;
```

```
  background-color: rgba(0, 0, 0, 0.95);
  transform: scale(0, 0);
  transition: transform 150ms linear;
  overflow: hidden;
}
```

This class defines the element that is displayed over the entire page, or commonly known as a *modal* window.

To position the element in the absolute center of the user's view, we set *position: fixed*, *top, left, right, and bottom to 0*, and *margin: auto*. This combination ensures that, no matter the size of the modal, it's center point will be at the center of the user's view.

Tip: Some designers recommend using ***100vw and 100vh*** instead of 100% - this should be avoided in proper UI design. The problem with *100vw* and *100vh* is the values are calculated based on the view of the entire device (for mobile; it's safe on desktops, but avoid it to remain consistent).

For example, 100vh would include the height of the document, plus the address bar of the web browser on your user's smartphone. This creates usability problems, because now part of your gallery's modal will be out of view to the user's device, unless they close the address bar, title bar, etc, and your document takes the entire height of the device (which is difficult and pointless to achieve = they may not want that).

transform: scale(0, 0); makes the element appear to be hidden. This is useful when creating *animations* that the user will experience.

display: block and *display: hidden* can't be animated, as they instruct the browser to recalculate the dimensions of the element, and reflow the display of surrounding elements (if applicable). It's slow, and doesn't animate at all.

You may be able to get away with *height: 0* and *width: 0*, to make an element appear hidden, but then you have to manage the height/width, and recalculate it when it's time to display a photo.

transform and other animation CSS, take advantage of the graphics engine running on the computer/device, and performs animations more smoothly than recalculating each dimension individually, and with each step.

#70 – 72: Show the Photo Frame

```
.modal-holder.shown {
  transform: scale(1,1);
}
```

We'll add this class to the modal when we display the photo to the user. It will then scale the modal to its full size with an *expanding* effect. When we close the modal, and remove the class, the transition will reverse, creating a *shrinking* effect.

#74 - #84: Photo Container

```
.modal-holder > .modal-content {
  position: absolute;
  width: calc(100% - 20px);
  height: calc(100% - 20px);
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
  z-index: 1;
}
```

The .modal-content class contains the photo the user is viewing. We set the *position: absolute, top, left, right, and bottom, to 0*, with *margin: auto*; to place it in the center of the *modal-holder* element.

Tip: When you have a parent element with position: relative, position: absolute, or position: fixed, you can position a child with an absolute location *relative* to the parent's position. In our code, we position .modal-content in the center, but the following would put it in the bottom-right of .modal-holder:

```
.modal-holder > .modal-content {
  position: absolute;
  width: calc(100% - 20px);
  height: calc(100% - 20px);
  right: 0;
  bottom: 0;
  margin: auto;
  z-index: 1;
}
```

width: calc(100% - 20px) and *height: calc(100% - 20px)* ensures the *.modal-content* has a 10px margin on each side. Which begs the question: why didn't I just set *margin*: *10px*? Answer: box-sizing doesn't include margins, and so the element would be 100% + 10px on each side. That wouldn't look so good, now would it?

Furthermore, if we used *padding: 10px*, it would put the visual part of the element still stretched to the edges of *.modal-holder*. What we want is, that even if we add a border, padding, or background-color to *.modal-content*, it will retain the separation from the edges of *modal-holder*.

#86 – 97: Full-Size Photo

```
.modal-holder > .modal-content > img {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
  max-width: 100%;
  max-height: 100%;
  width: auto;
  height: auto;
}
```

This positions the full-size photo in the center of *modal-content*, and ultimately in the center of the user's screen / window, right where they expect it. As with the thumbnails, the width, height, max-width, and max-height values are set to keep the photo from being larger than the viewable area.

#99 – 110: Gallery Controls

```
.modal-holder > .modal-controls {
  position: absolute;
  width: 100%;
  height: 50px;
  left: 0;
  right: 0;
  bottom: 0;
  z-index: 2;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  flex-flow: row wrap;
}
```

The *.modal-controls* class displays buttons for the user to control the full-size gallery, in our case, previous and next buttons.

#112 – 120: Gallery Buttons

```
.modal-holder > .modal-controls > a {
   flex: 1;
   font-size: 24px;
   color: #FFFFFF;
   text-align: center;
   line-height: 50px;
   height: 50px;
   text-decoration: none;
}
```

This styles the individual buttons. We use anchors, again, to allow users to tab through with a keyboard. Setting the *line-height* and *height* to the same value vertically centers the single-line of text, which is our arrow icon from FrontAwesome.

#122 – 134: Close Button

```
.modal-holder > a.modal-close {
   position: absolute;
   top: 5px;
   right: 5px;
   width: 50px;
   height: 50px;
   font-size: 24px;
   color: #FFFFFF;
   text-align: center;
   line-height: 50px;
   text-decoration: none;
   z-index: 3;
}
```

The user can use this button to close the *modal-holder* and return to the gallery thumbnails. We place it in the upper-right to keep it away from the next/previous buttons, so the user doesn't accidentally close the photo.

Section 4: JavaScript – gallery.js

Even without JavaScript, using anchors in our gallery gives non-JavaScript users the ability to click on the anchor and see the full-size photo – though it lacks the fancy (simple) interface we're building.

Here, you'll learn how to preload images, populate the list of photos, and why creating elements on the fly is smarter than searching for elements by ID, or (worse) having markup within your JavaScript.

Content of gallery.js:

```
1. (function($) {
2. if(typeof($) != "function")
3.     return;
4.
5. /**
6. * PhotoGallery class
7. */
8. function PhotoGallery(container) {
9.
10.    //gallery element
11.    this.Container = container;
12.
13.    //Our Photo URLs
14.    this.Photos = [];
15.
16.    //Starting Index
17.    this.PhotoIndex = -1;
18.
19.    /*
20.    * Photo Images
21.    * Create three, so we can pre-load the previous/next
22.    * photos for faster loading on the user's experience.
23.    */
24.    this.PhotoCurrent = $(document.createElement("img"));
25.    this.PhotoPrevious = $(document.createElement("img"));
26.    this.PhotoNext = $(document.createElement("img"));
27.
28.    //Show next photo when user clicks the current photo
29.    this.PhotoCurrent.on('click', this.next.bind(this));
30.
31.    //Modal Container
32.    this.ModalHolder = $(document.createElement("div"));
33.    this.ModalHolder.addClass('modal-holder');
34.
35.    //Modal Content - Holds the current photo.
36.    this.ModalContent = $(document.createElement("div"));
37.    this.ModalContent.addClass('modal-content');
38.    this.ModalContent.append(this.PhotoCurrent);
39.    this.ModalHolder.append(this.ModalContent);
40.
41.    //Modal controls - navigation, share buttons, etc.
42.    this.ModalControls = $(document.createElement("div"));
43.    this.ModalControls.addClass('modal-controls');
44.    this.ModalHolder.append( this.ModalControls );
45.
46.    //Close Button
47.    this.ButtonClose = $(document.createElement("a"));
48.    this.ButtonClose.addClass('modal-close fas fa-times-circle');
49.    this.ButtonClose.attr('href', '#');
50.    this.ModalHolder.append(this.ButtonClose);
51.    this.ButtonClose.on('click', function(ev) {
```

```
52.      ev.preventDefault();
53.      this.close();
54.    }.bind(this));
55.
56.    //Previous Button
57.    this.ButtonPrev = $(document.createElement("a"));
58.    this.ButtonPrev.addClass('fas fa-arrow-left');
59.    this.ButtonPrev.attr('href','#');
60.    this.ModalControls.append(this.ButtonPrev);
61.    this.ButtonPrev.on('click', function(ev) {
62.      ev.preventDefault();
63.      this.prev();
64.    }.bind(this));
65.
66.    //Next button
67.    this.ButtonNext = $(document.createElement("a"));
68.    this.ButtonNext.addClass('fas fa-arrow-right');
69.    this.ButtonNext.attr('href','#');
70.    this.ModalControls.append(this.ButtonNext);
71.    this.ButtonNext.on('click', function(ev) {
72.      ev.preventDefault();
73.      this.next();
74.    }.bind(this));
75.
76.    //Keyboard handler
77.    $(window).on('keyup', this.handleKeyUp.bind(this));
78.
79.    //prepare our photos
80.    this.loadPhotos();
81.
82.    //Add the gallery modal to the document
83.    $("body").append(this.ModalHolder);
84. };
85.
86. /*
87. * Loads the photos URLs to the gallery.
88. */
89. PhotoGallery.prototype.loadPhotos = function() {
90.    this.Photos.length = 0;
91.    this.PhotoIndex = -1;
92.    let i = 0;
93.    let that = this;
94.    this.Container.find('a').each(function() {
95.      $(this).data('photo-index', i);
96.      $(this).on('click', function(ev) {
97.        ev.preventDefault();
98.        ev.stopPropagation();
99.        that.show($(this).data('photo-index'));
100.       $(this).blur();
101.     });
102.     that.Photos.push($(this).attr('href'));
103.     i++;
104.   });
105.};
106.
107./*
108.* Handles keyboard commands.
109.* Commands are ignored if the modal is not displayed.
110.*/
111.PhotoGallery.prototype.handleKeyUp = function(ev) {
112.   if(!this.ModalHolder.hasClass('shown'))
113.     return;
114.
```

```
115.   //prevent default on known controls only
116.   switch(ev.keyCode) {
117.     case 37 :
118.     case 38 :
119.     case 39 :
120.     case 40 :
121.     case 33 :
122.     case 34 :
123.     case 32 :
124.     case 36 :
125.     case 35 :
126.     case 27 :
127.       ev.preventDefault();
128.       ev.stopPropagation();
129.       break;
130.
131.     default :
132.       return;
133.       break;
134.   }
135.
136.   switch(ev.keyCode) {
137.     //escape - close
138.     case 27 :
139.       this.close();
140.       break;
141.
142.     //Next (space, right, down, pgdown)
143.     case 32 :
144.     case 40 :
145.     case 39 :
146.     case 34 :
147.       this.next();
148.       break;
149.
150.     //Previous (left, up, pgup)
151.     case 33 :
152.     case 37 :
153.     case 38 :
154.       this.prev();
155.       break;
156.
157.     //End - last Photo
158.     case 35 :
159.       this.show(this.Photos.length - 1);
160.       break;
161.
162.     //Home - First Photo
163.     case 36 :
164.       this.show(0);
165.       break;
166.   }
167.};
168.
169./**
170.* Closes the gallery's modal element.
171.*/
172.PhotoGallery.prototype.close = function() {
173.   this.ModalHolder.removeClass('shown');
174.};
175.
176./**
177.* Displays the next photo.
```

```
178. */
179. PhotoGallery.prototype.next = function() {
180.   this.show(this.PhotoIndex + 1);
181. };
182.
183. /**
184. * Displays the previous photo.
185. */
186. PhotoGallery.prototype.prev = function() {
187.   this.show(this.PhotoIndex - 1);
188. };
189.
190. /**
191. * Displays the photo at the given index.
192. * @param {Number} index
193. */
194. PhotoGallery.prototype.show = function(index) {
195.   //keep within bounds of the Photos
196.   if(index < 0)
197.     index = this.Photos.length - 1;
198.   else if(index >= this.Photos.length)
199.     index = 0;
200.
201.   if(index < 0)
202.     index = 0;
203.
204.   //ignore if photo does not exist
205.   if(!this.Photos[index])
206.     return;
207.
208.   //set current photo, show modal
209.   this.PhotoCurrent.attr('src', this.Photos[index]);
210.   this.ModalHolder.addClass('shown');
211.   this.PhotoIndex = index;
212.
213.   //load next photo
214.   var ni = index + 1;
215.   if(ni >= this.Photos.length)
216.     ni = 0;
217.
218.   if(this.Photos[ni]) {
219.     this.PhotoNext.attr('src', this.Photos[ni]);
220.   }
221.
222.   //load previous photo
223.   var pi = index - 1;
224.   if(pi < 0)
225.     pi = this.Photos.length - 1;
226.
227.   if(this.Photos[pi]) {
228.     this.PhotoPrevious.attr('src', this.Photos[pi]);
229.   }
230. };
231.
232. $(document).ready(function() {
233.   $(".gallery").each(function() {
234.     var gallery = new PhotoGallery($(this));
235.   });
236. });
237.
238. }(jQuery));
```

Let's look at the lines of code in more detail, so you know what's going on and why.

#1 - #238: Anonymous Function

```
(function($) {
   //the rest of our code
} ( jQuery ) );
```

We wrap all of our JavaScript in an anonymous function. This helps reduce the chance that anything we write will interfere with other scripts on the web page.

#2-3: jQuery validation

```
if( typeof($) != "function")
  return;
```

jQuery must be added to the document *before* our script. Here, we check if it hasn't been added, and ignore the rest of our code if not.

#8 - #85: PhotoGallery class.

```
function PhotoGallery(container) {
   //lines 9 – 84
}
```

This is how you declare *classes* in JavaScript: they are *functions*. (ECMAScript 6 allows for a *class* wrapper syntax that handles this in the background.)

#11: Gallery Container

```
this.Container = container;
```

We create a *PhotoGallery* for each *.gallery* element on the page, and pass the element as the *container*. We'll use this later to find the URLs for each photo.

#17: Current Photo Index

```
this.PhotoIndex = -1;
```

This keeps track of the current photo shown to the user. We use this in the *next()* and *prev()* functions.

#24-26: Photo holders.

```
this.PhotoCurrent = $(document.createElement("img"));
this.PhotoPrevious = $(document.createElement("img"));
this.PhotoNext = $(document.createElement("img"));
```

PhotoCurrent is the photo the user views.
PhotoPrevious is the photo that is loaded in preparation for when the user clicks *ButtonPrev*.
PhotoNext is the photo that is loaded in preparation for when the user clicks the *ButtonNext*.

Preloading images allows us to improve the user's experience, appearing to reduce the time needed to load a photo. Since the photo will already be loaded in *PhotoPrevious* / *PhotoNext*, the browser/device can pull the photo from the cache.

#29: Current Photo Interaction

```
this.PhotoCurrent.on('click', this.next.bind(this));
```

When the user clicks the full-size photo, we'll advance the gallery to the next photo.

#32 and #33: Modal Holder

```
this.ModalHolder = $(document.createElement("div"));
this.ModalHolder.addClass('modal-holder');
```

This is the container for the modal, or the large black background that is overlayed on the web page.

#36 - #39: Modal Content

```
this.ModalContent = $(document.createElement("div"));
this.ModalContent.addClass('modal-content');
this.ModalContent.append( this.PhotoCurrent );
this.ModalHolder.append( this.ModalContent );
```

This is the container that is positioned in the center of the screen, and will contain *PhotoCurrent*.

#42 - #44: Modal Controls

```
this.ModalControls = $(document.createElement("div"));
this.ModalControls.addClass('modal-controls');
this.ModalHolder.append( this.ModalControls );
```

This is where we put the *ButtonPrev* and *ButtonNext* buttons.

#47 – 54: Close Button

```
this.ButtonClose = $(document.createElement("a"));
this.ButtonClose.addClass('modal-close fas fa-times-circle');
this.ButtonClose.attr('href', '#');
this.ModalHolder.append(this.ButtonClose);
this.ButtonClose.on('click', function(ev) {
  ev.preventDefault();
  this.close();
}.bind(this));
```

ButtonClose closes the large photo display. When we call *close()*, we remove the .shown class from *.modal-holder*, and it reverts to its original scaled-down size: transform: scale(0,0), along with a shrinking animation.

We use anchors for our buttons, instead of *buttons* or *images*, so they can be focused by users who don't have access to a mouse, or prefer to use the keyboard to navigate.

#57 – 64: Previous Button

```
this.ButtonPrev = $(document.createElement("a"));
this.ButtonPrev.addClass('fas fa-arrow-left');
this.ButtonPrev.attr('href', '#');
this.ModalControls.append(this.ButtonPrev);
this.ButtonPrev.on('click', function(ev) {
  ev.preventDefault();
  this.prev();
}.bind(this));
```

ButtonPrev displays the previous photo. If we're at the first photo in the gallery, it displays the *last* photo, and moves backward from there.

#67 – 74: Next Button

```
this.ButtonNext = $(document.createElement("a"));
this.ButtonNext.addClass('fas fa-arrow-right');
this.ButtonNext.attr('href', '#');
this.ModalControls.append( this.ButtonNext );
this.ButtonNext.on('click', function(ev) {
  ev.preventDefault();
  this.next();
}.bind(this) );
```

ButtonNext advances the gallery to the next photo. If we get to the last photo, and the user clicks ButtonNext, we move back to the *first photo* in the gallery, and keep moving forward from there.

#77: Keyboard Handler

```
$(window).on('keyup', this.handleKeyUp.bind(this));
```

Bind an event handler for when the user presses keys on their keyboard. We'll ignore keyboard commands when the modal isn't visible.

#89: Load Photos

```
PhotoGallery.prototype.loadPhotos()
```

This function loads the URLs of the photos into the Photos array, assigns an index to each photo, and adds an event handler for when the user clicks the thumbnail.

#95: Photo Index

```
$(this).data('photo-index', i);
```

We assign an index to each thumbnail anchor, so when the user clicks on it, we can display the photo they clicked on.

#96 - #100: Thumbnail Event Handler

```
$(this).on('click', function(ev) {
  ev.preventDefault();
  ev.stopPropagation();
  that.show( $(this).data('photo-index') );
  $(this).blur();
});
```

This is the event handler attached to the thumbnail. We use anchors to allow users with keyboards to tab through the photo thumbnails, as *img* elements cannot retain focus and instruct screen readers what they are linking to.

Unlike our other anchor event handlers, we allow this one to retain scope of *this* to reference the anchor. Then, we create a *that* variable, to reference the *PhotoGallery*.

```
let that = this;
```

This is done because, while the event object, *ev*, contains a *target* property, that *target* property may reference the *img* element. Since the index of the photo is stored on the anchor, we need to pull it from there.

#111 – 167: Keyboard Handler – *handleKeyUp(ev)*

```
(See full code above, lines 111 through 167).
```

This handler performs three steps:

1. If *ModalHolder* is not displayed, then we allow the default keyboard behavior continue.
2. The first switch detects the keyCode that we want to control. If the keyCode is not recognized, we allow the default behavior of the keyboard.
3. The second switch responds to the keycodes we recognize:
    - *Escape*: Close the photo gallery.
    - *Space, Right, Down, PageDown*: Next photo.
    - *Left, Up, Page Up*: Previous Photo
    - *End*: Last Photo
    - *Home*: First Photo

#195 – 232: Show the Photo – *show(index)*

```
(See full code above, lines 195 through 232).
```

This function displays the full-size photo to the user, and preloads next/previous photos. The single parameter, *index*, is checked for validity, keeping it within bounds of the *Photos* array, and creates the cycling effect when clicking next on the last photo.

Remember, pre-loading the image provides a better user experience, so the next and previous photos pull from memory / cache. This provides the illusion that the previous/next photos have loaded faster.

#234 – 238: Create the galleries.

We create the galleries only after all HTML, JavaScript, and CSS files, have loaded.

Tip: Just like it's good to wrap your code in an anonymous function, it's also good to delay execution of *most* code until the document has loaded. (This doesn't mean all images, video, and audio, have to load; just the HTML, JavaScript, and CSS files.)

The End

Seriously, the end.

There’s no more to this guide.