# Part 0: Get the code

1. Download the Project 2 ZIP file from Google Classroom.
2. Move the contents of the ZIP file to your computer’s local directory.
3. Using your GitHub account, create a new project using your Project 2 directory from Step 2. I recommend using the following project name: lastname-p2
4. Connect your GitHub repository to a new Heroku app. I recommend using the following name:  
   lastname-p2.herokuapp.com
5. Follow the directions below. Please commit to your GitHub repository often to keep track of your code and to show your progress.
   1. When committing changes to GitHub, use the “best practices for commit messages” (available on [here](https://chris.beams.io/posts/git-commit/)).

\**note*: you do not have to edit *index.html* or *index.php* at all to complete this project (but you may if you want). The primary files you will be editing for this project are *gallery.js* and *style.css*

*\*\* note: one of the requirements for this project is to COMMENT YOUR CODE. Which means that wherever you can in your JavaScript file(s), you should be explaining your code using comments: //*

# Part 1: Create a JavaScript Object

1. Create a Javascript Object, called **GalleryImage,** by using a function[[1]](#footnote-1) which contains the following properties:
   1. String : location (the location the photo was taken)
   2. String : description (a description of the photo)
   3. String : date (the date the photo was taken)
   4. String : img (the src URL for the photo)

# Part 2: Slide Show

1. Use an XMLHttpRequest[[2]](#footnote-2) called **mRequest** to fetch the JSON located inside the *images.json* file. This will be your default JSON file.
2. Create a JSON object that contains the retrieved JSON[[3]](#footnote-3) string (in this case, a list of photo URLs and related metadata).
3. Iterate through the JSON object and create **GalleryImage** objects using JavaScript for each image in the JSON object. Put each of these new GalleryImage objects into an array called mImages[].
   * + 1. Get the src URL from the JSON file and put it into your GalleryImage object.
4. Create a slide show to iterate over all the **GalleryImage** Objects in your mImages[] array. The *gallery.js* file includes code that creates and continuously runs a timer. When the timer goes off after 5000ms (5 seconds), it calls swapPhoto().
   1. Implement swapPhoto() so when it gets called, it finds the <img> inside the **div#slideShow** and swaps its current src with a new src on a **GalleryImage** object in your array. Use the **GalleryImage’s** metadata information (Description, Location, and Date) and update the **div.details** section with the extracted info.
   2. Use a counter called **mCurrentIndex** to loop through all **GalleryImage** objects in the array. When you reach the end, start over again.
   3. The timer code is intelligent enough to not run when your webpage isn't currently focused. Keep that in mind when using the console. You could have the console open and not see events happening due to the webpage being unfocused (i.e. the current browser tab is not visible).

# Part 3: Gallery

1. Change the *img* element with a class value = “*moreIndicator rot90*” to be horizontally centered and bottom aligned with its sibling *img* element.  
   (http://stackoverflow.com/questions/14233341/how-can-i-rotate-an-html-div-90-degrees)
2. Use CSS to implement the *.rot90* and *.rot270* selectors that will transform selected elements to rotate 90 or 270degs. Use the CSS transition property to create a nice animation.  
   (http://robertnyman.com/css3/css-transitions/css-transitions-rotation.html)
3. Add a click handler to the **img.moreIndicator** that does the following:
   1. Add a class attribute value = “*rot270*” if the element currently has a class value with “rot90”; else remove the “*rot270*” class and add “*rot90*”. This will cause the arrow to animate upside down. (https://api.jquery.com/hasclass/)
   2. Slides down/up the **div.details** depending on the arrow direction. (http://api.jquery.com/fadetoggle/)
4. Use jQuery to offset the **#nextPhoto** image so that it is flush with the right side of the **#gallery** div (see *info/finished\_gallery.png* for visual).
5. Add hover handlers (in CSS) to the **#nextPhoto** and **#prevPhoto** in the **div#nav** so that when the mouse pointer goes over them, their opacity is set to .8 (80%).
6. Make sure that all displayed photos are centered in the container, and that they fit entirely within the container (i.e., photos should not extend past the container if they’re too big).
7. Add click handlers (in jQuery) to the **#nextPhoto** and **#prevPhoto** in the **div#nav** so that when they are clicked, they will go to the next photo or the previous photo, respectively. They should be shown in the array order. On the last photo, it should loop back to the first photo when **#nextPhoto** is pressed. Likewise, on the first photo, it should loop back to the last photo when **#prevphoto** is pressed.

# Part 4: Alternate JSON input using GET

1. Using Javascript/jQuery, add a GET handler that accepts a variable called “json” that allows a user to point to an alternate JSON file. If the alternate JSON file is valid, use that file as the gallery’s input. If there is no json file provided, or the provided JSON file is invalid, use the default *images.json* file. For example, the following should work when you put it into your browser: index.html?json=images-short.json

# Part 5: Create your own JSON

1. Create your own JSON file based on *images.json* (please name it *extra.json*) that has URLs and metadata for your own photo/image collection.
2. There should be a minimum of 5 images used in the JSON.
3. Make sure that *index.html?json=extra.json* (see requirements from Part 4) show your images and metadata in your gallery.

# Part 6: Deploy onto the Cloud

1. Make sure your github repository is connected to a heroku app (or any other online-accessible web server you prefer).
2. Deploy your website. If you are using Heroku, here is how to resolve three common errors:
   1. Your website won’t deploy; it’s likely you forgot to include an index.php file. Heroku needs to have that file to determine you want a PHP-compatible server. Use the one included in the class zip file (which automatically links you to index.html).
   2. Your website doesn’t look correct; it’s not loading some files. Make sure you don’t have a /vendor/ or /vendors/ folder in your root directory. You will have to rename this folder to something like /external/ (and update all your code to reflect this change from /vendors/ to /external/), since this is a protected/special directory used internally by Heroku.
   3. Your website doesn’t look correct; it’s not loading some files. This usually happens because you are loading secure (https) and not-secure (http) files together. A simple solution is to use the http address and submit that URL. For example:   
      *https://myapp-p2.herokuapp.com*, submit *http://myapp-p2.herokuapp.com*  
      (notice the missing “s”)

# Take one Part 7: Submit

1. Take one screenshot of your working website with the details section opened.
2. Submit the following on Google Classrom by the due date:
   1. A link to your project’s Github repository  
      (example: <http://www.github.com/mjslee/is219s16lee-p2)>
   2. A link to your project’s Heroku server/website  
      (example: <http://www.is219s16lee-p2.herokuapp.com)>

|  |  |  |  |
| --- | --- | --- | --- |
| **Project 2 – Grading Rubric** | | **Points Possible** | **Your Points** |
| 1 | Did you properly use Github to track your changes (with appropriate commit messages)? | 8 |  |
| 2 | Did you comment all your Javascript code, explaining what the code does? | 7 |  |
| 3 | Did you properly deploy your project to a Heroku (or other web) server? | 4 |  |
| 4 | Is XMLHttpRequest correctly implemented to read a default JSON file (i.e. images.json)? | 7 |  |
| 5 | Does the script correctly create and store GalleryImage objects into an array called mImages? | 7 |  |
| 6 | Does script automatically advance the slides every 5 seconds? (this is already implemented for you) | 2 |  |
| 7 | Does calling swapPhoto() show the next image (and its details) from mImages on the screen? | 5 |  |
| 8 | Does swapPhoto() automatically go back to the first photo after it reaches the last one in the mImages array? | 4 |  |
| 9 | Does the current photo’s metadata (location, description, date) show up when the moreIndicator button is pressed? | 5 |  |
| 10 | Does the moreIndicator button animate (flip) when it’s pressed? | 3 |  |
| 11 | Is the moreIndicator button center-aligned and overlaid on the bottom of the displayed photo? | 3 |  |
| 12 | Are the “previous” button and “next” buttons aligned properly (on either end)? | 3 |  |
| 13 | Are the “previous” button and “next” buttons pointing in the correct direction? | 3 |  |
| 14 | Do the “previous” button and “next” buttons change opacity when the mouse goes over them? | 3 |  |
| 15 | Does the “previous” button go the previous image? | 3 |  |
| 16 | Are all the displayed photos centered and fit within the container? | 3 |  |
| 17 | When on the first photo in the mImages array, does the “previous” button go back to the last image in the mImages array? (i.e. loop around) | 5 |  |
| 18 | Does the “next” button go the next image? | 3 |  |
| 19 | When on the last photo in the mImages array, does the “next” button go back to the first image in the mImages array? (i.e. loop around) | 4 |  |
| 20 | Is the GET correctly implemented so that I can load an alternate .JSON file? | 8 |  |
| 21 | Did you make (and can I load) your *extra.json* file? | 5 |  |
| 22 | Does your *extra.json* file display correctly with at least 5 images with different metadata? | 5 |  |
| 23 | DEDUCTION: -10 per day late. | -10/day |  |

1. http://www.phpied.com/3-ways-to-define-a-javascript-class/ [↑](#footnote-ref-1)
2. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using\_XMLHttpRequest [↑](#footnote-ref-2)
3. http://www.w3schools.com/json [↑](#footnote-ref-3)