Build an Interactive Web Page with Image Filters

Duke University
Programming Foundations with JavaScript, HTML and CSS
Module 5: MiniProject: Image Filters on the Web

Congratulations! You are ready to implement what you have learned in this course in the culminating MiniProject. In this assignment, you will build an interactive web page where a user can upload an image and apply filters to it, including at least one filter designed by you.

When you are finished, someone visiting your page should be able to:

- Upload and display an image
- Apply at least three filters to the image
- Reset the image to its original version

Part 1: Make a Prototype

As you learned in the Event-Driven Programming lesson, it is useful to make a prototype web page as the first step toward a sophisticated interactive web page.

For this MiniProject, start by creating the layout for your web page:

1. Add your HTML elements. Specially, your prototype page should have:

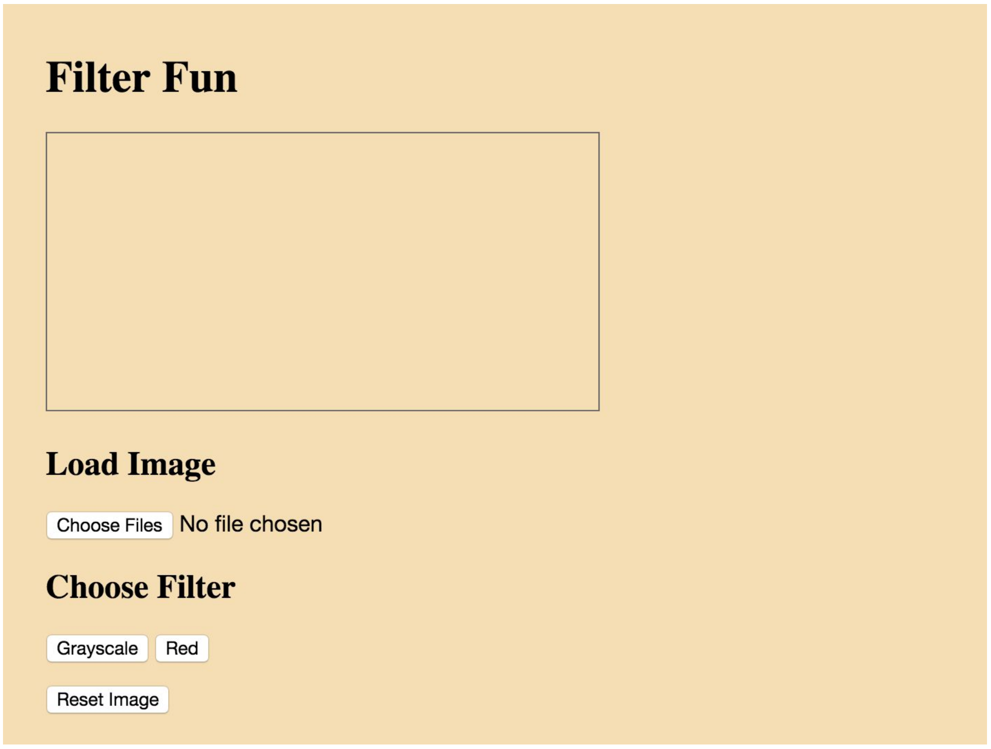
- A heading and subheadings
- A canvas
- A file input element, which only allows the user to upload a single image file
- At least 3 buttons, one for each image filter you will create
- A fourth button to "reset" the image (i.e., show the original/unfiltered image)

2. In the JavaScript panel, create at least 5 functions, one for each of your input elements. Be sure to name the functions clearly (e.g., if a function is supposed to load an image to the canvas, perhaps name the function **loadImage()**). For the time being just have the functions alert the user that the button or file input has been used.

3. Have the event handlers for each of your input elements call the appropriate function (e.g., onclick, onchange, etc.).

4. Be sure to use CSS to style your web page in a way that appeals to you. Experiment with margins and padding, and find new colors and font styles to make the page look great! Check out W3School's resources for CSS: http://www.w3schools.com/css/.

Here is a screenshot of an example prototype page. Feel free to structure and style yours however you want. Just make sure you have the canvas and input elements described above.



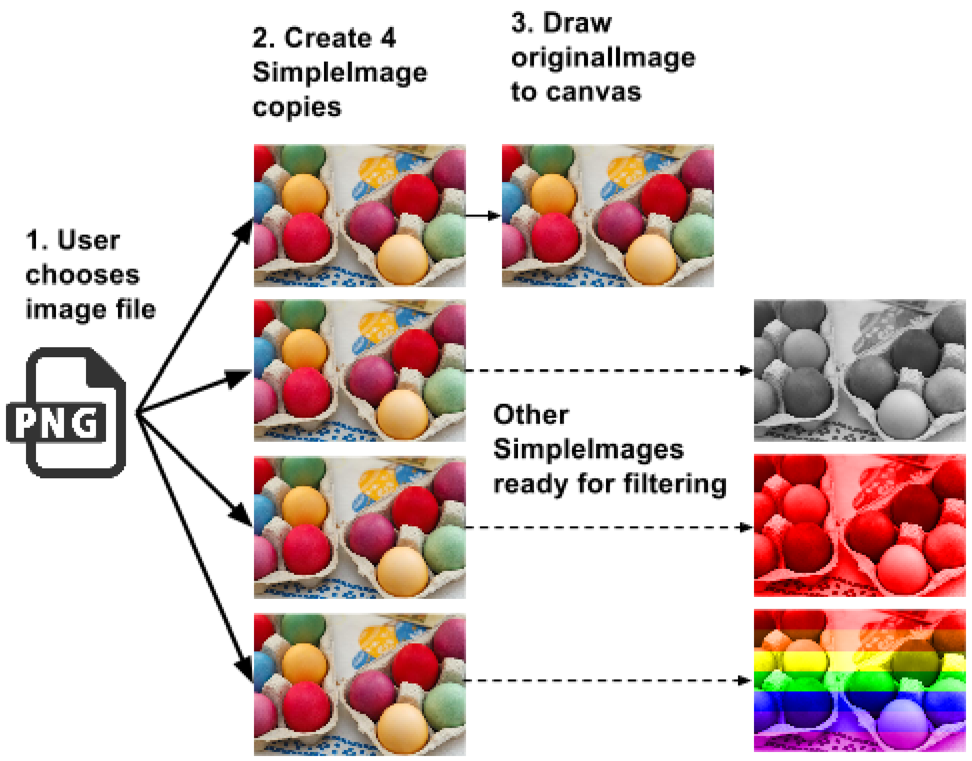
Part 2: Grayscale Filter

Plan your code

Let’s begin to think through how the JavaScript code should be organized. When the user clicks the “Choose Files” button and selects an image file, your web page will create a SimpleImage variable of the image file and draw it to the canvas. Then, your user can apply any one of your filters to the image.

But what happens if the user applies one filter and then wants to switch to another filter? How can you let the user use one filter and then return to the original image to apply another filter? One way to solve this problem is to have your image loading function create a copy of the image for each filter the user could click. That way, you can manipulate one version of the image, while preserving a copy of the original image.

The image below demonstrates the planning for a page that has three filters but needs to retain a copy of the original image.


2. Create 4 SimpleImage copies
3. Draw originalImage to canvas
1. User chooses image file
PNG
Other SimpleImages ready for filtering

Since you will want to load the images in one function and process them in another, it is convenient to make them global variables. You may also want to initialize them to null, so that you can check if they have loaded later. Since you will make changes to the canvas element from multiple functions, it is convenient to make this a global variable too.

Another programming decision you will need to make is what the reset button should do. Should it display the original image? Should it also reset all of the image variables to the original image?

Complete image loading function

Implement your image loading function so that when the Choose Files button is clicked, the image is uploaded and displayed on the canvas. You may also want to set your global image variables to copies of the original image at this point.

Implement the grayscale filter

You have practiced implementing grayscale on a web page before. In this version, we suggest you make two changes to your code organization, but the main algorithm will be the same. The changes are:

1. Check that the copy of the image for this filter has loaded before you apply the filter to it. It is convenient to use a helper function for this that takes an image as a parameter and returns true if the image is complete and not null, and otherwise alerts the user the image is not loaded and returns false. Refer to your green screen code for an example of this kind of check and write this helper function now.
2. Put the actual image modifying steps in a helper function. This means you can separate the image modification from the image loading, the check, and displaying the final image on the canvas.

An example of the function triggered by the grayscale button could be:

```
function doGray() {
  if ( imageIsLoaded(grayImage) ) {   ← check if image is loaded
    filterGray();                     ← function performs the grayscale work
    grayImage.drawTo(canvas);         ← display image
  }
}
```

This function calls the helper function `imageIsLoaded()`, which will return either true or false, then calls the helper function that does the grayscale algorithm, then draws the grayscale image to the canvas.

Make sure to test that your grayscale filter is applied when you click the grayscale button! Also check that your reset button displays the original image, and that your code alerts you if you click the grayscale filter button before an image has been loaded.

Complete reset button

Write a reset function that will be called by your reset button's event handler. The function should:

- Check if the original image is loaded.
- If the image is loaded: display it on the canvas, and reset all of the global variables for filter images to the original image. This way, if you run one of your filters after resetting, the filter will run on the original image, not on the already filtered image.

Need help? Refer to a previous Try It! reading, such as "Convert an Image to Grayscale" or "Green Screen Online." You can also seek help in the forums!

Part 3: Red Filter

Translate red filter algorithm to code

Now that you have implemented the grayscale image filter, it is time to try an unfamiliar image filter algorithm: a red hue filter. Below is one possible algorithm for the red filter.

For each pixel in the image:

- **Calculate the average of the pixel's RGB values**
- **If the average is less than 128**: set the pixel's red value to two times the average, set the pixel's green value to zero, and set the pixel's blue value to zero.
- **Otherwise**: set the pixel's red value to 255, set the pixel's green value to to two times the average minus 255, and set the pixel's blue value to to two times the average minus 255.

Using the DukeLearnToProgram JavaScript environment, translate the above red filter algorithm into JavaScript. Test with different images to make sure that your code is working correctly. What types of images would be good to test the code with?

If you would like to understand the logic behind this algorithm, see this document explaining the red filter.

Add red filter to web page

Once you have implemented the red filter in DLTP, add it to your web page. We suggest using the same approach as with the grayscale filter: have one function that actually modifies the image (in the grayscale example, this function was `makeGray`), then have another function (in the grayscale example, this function was `doGray`) that checks that the image is loaded, calls the function that modifies the image, and displays the filter on the canvas.

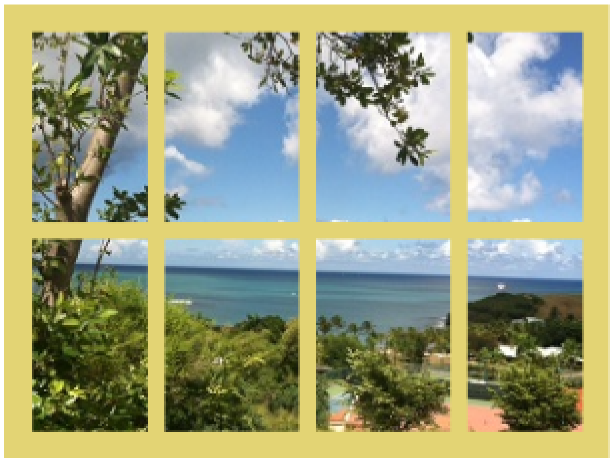
Try your red filter button to see if it works! Try pressing your red filter button several times. What happens to the image? What happens if you now click the grayscale filter button, then the red filter button?

This is a great time to check whether your reset button works properly and resets all the filter images to the original image, so that the next time you run the filters they run on the original image, not on the already filtered image.

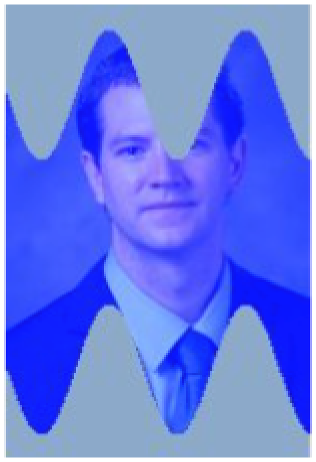
Part 4: Design your own filter

As the final part of your MiniProject, design and program your own image filter. It can do whatever you want, but it should be something original, not one of the filters you have seen in the class. Here are a couple of examples.

Window Pane



Trigonometry Shark



In these examples, pixels are manipulated based on where they are in the image, as well as their RGB values. You can can also include more complex math operations with JavaScript's Math object methods: http://www.w3schools.com/js/js_math.asp.

Be creative!

Once you have decided what you want your filter to do, you should follow the seven step process:

1. Work some small examples (i.e., a few to several pixels) by hand.
2. Write down the steps you took. Be very specific—this will help in translating to code later.
3. Look for patterns and generalize your solution.
4. Check your algorithm by hand on a few small examples to make sure you have not missed anything.
5. Once you are satisfied that your algorithm does what you want it to, translate it to code. You should develop your code in the Duke environment on the course site: http://www.dukelearntoprogram.com/course1/example/index.php (linked in the **Resources** tab as well). Later you will put this code on your web page.
6. Test your code. Think of what types of images would be good to try your filter on.
7. Debug any problems in your code.

Once your code is working in the Duke environment, add it to your web page as you added the grayscale and red filters and test it by uploading a few different images to your web page.

Part 5: Personalize and share

The appearance of your web page is up to you! You should think about how you want to structure and style your web page using the HTML and CSS you have learned.

Once you have a version you would like to share, post a link to your pen in the forum “Share Your Success.” Check out some of the other learners’ projects, and give them your feedback!

Need help? Review the documentation at http://www.dukelearntoprogram.com/course1/doc/ to remind yourself of the SimpleImage methods and common HTML and CSS for laying out and styling a web page.