This section includes two problems that require working with images and one working with histograms. All three are directly applicable to the ImageShop project. The first gives you some practice working with pixel values and looping over an image and the second with reassigning pixel values and creating a new empty image. You'll need the third problem for Milestone 5 in the ImageShop project.

1. **Red-eye Correction**

   When you take a picture of a face, the light from the flash often reflects off the blood vessels in the retina, which makes the pupils of the eye appear bright red. This phenomenon is called the *red-eye effect* and is illustrated in the following photograph:

   Modern image-processing software includes sophisticated algorithms to eliminate the red-eye effect, but you can do a reasonable job using much simpler strategies similar to those required for Milestone 4, in which you have to recognized pixels in which the green component dominates.

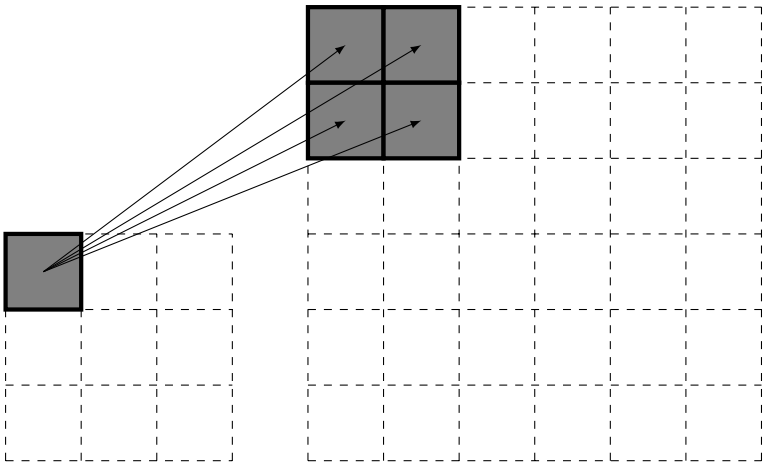
   For this problem, your job is to implement a **Red Eye** button that goes through every pixel in the image and finds those in which the red component is at least twice as large as the maximum of the green and blue components. When it finds a pixel that meets this criterion, your code for the **Red Eye** button should replace it by a new pixel in which the red component is lowered to equal the maximum of the green and blue components.

   If you apply this transformation to the above cat picture, you get the following image:

2. **Image doubling**

   A common transformation that is desired in pictures is that of *scaling*: making the image larger or smaller. Scaling an image increases or decreases the number of pixels that make up the image, and thus decisions need to be made about what pixels to throw away (if scaling down) or how to generate extra pixels (if scaling up).

   If scaling up by an integer multiple, one technique is to just copy the same pixel value from the smaller image into multiple pixels in the new, larger image. If the image is doubling in size, this means copying each pixel from the smaller image into 4 pixels in the larger image, forming a small 2×2 grouping, as indicated in the below diagram:



   Implement a **Double** button that doubles the size of the image on the screen. Doing so will require:

   - Creating a new, empty array of the desired dimensions
   - For each pixel in the original array, copy that pixel value to the 4 corresponding locations in the double image array
   - Return a `GImage` of the new pixel array

   Scaling an image by a non-integer multiple is significantly more complicated in deciding how to map the original pixels onto the new pixel grid without stretching or distorting the image.

3. **Cumulative Histograms**

A *cumulative histogram* represents a running sum of a histogram array, wherein the entry in each index represents the number of elements that are that size *or smaller* in the original data. Generating a cumulative histogram is particularly important for Milestone 5 in ImageShop in order to get the contrast stretched correctly.

Define a function `create_cumulative_histogram(hist)` which takes as an argument a histogram array, such as that generated by `create_histogram_array` from Problem Set 5. Your function should then return the corresponding cumulative histogram. As an example, if you started with

```
SCORES = [ 8, 7, 8, 6, 8, 10, 4, 9, 6, 9, 7, 8 ]
histogram_array = create_histogram_array(11, SCORES)
```

then the `histogram_array` should equal

0	0	0	0	1	0	2	2	4	2	1
0	1	2	3	4	5	6	7	8	9	10

and thus the returned output of `create_cumulative_histogram` would be:

0	0	0	0	1	1	3	5	9	11	12
0	1	2	3	4	5	6	7	8	9	10

While certainly not necessary, could you write the body of this function in a single line? Think about the advantages and disadvantages of doing so, both in terms of program readability and efficiency.

4. **Optional: Image Overlay**

   Picture editors lie Photoshop allow you to combine an image with an overlay to produce a variety of special effects. Commercial applications implement this opeartion in several different ways. For this problem, your job is to add an **Overlay** button that creates a new picture by interleaving the pixels from the current image and a new image loaded from file. Pixels in which the sum of the row and column indices are even should come from the current picture; those for which that sum is odd should come from the newly loaded picture.

   If you combine the image of Van Gogh's Starry Night with the view of Earth from Apollo 17, you get the following interleaved picture: