Lab Exercise 11: Pair Programming Loops Within Loops

Goals

Develop more experience working together in Processing, and to apply the power of for-loops. You will learn more about using loops within loops. And, you will practice programming with a partner, as outlined in Homework 15.

Reminder About Placing A Image On Your Page

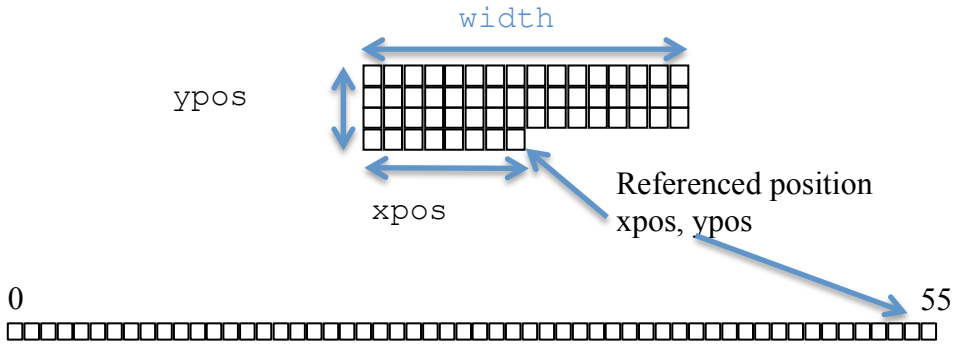
Recall the following points from Assignment 12 and Lab 09:

- Photos need to be in the same folder as the .pde file; jpg, .png and .gif are OK.
- One or more `PImage` variables, like `myImage`, must be declared.
- For us, it is much easier if the canvas size exactly matches the size of the photo.
- Photos need to be *loaded* into the program and assigned a name as in
  `myImage = loadImage("greatPic.jpg");`
- The photo must be placed on the canvas at a specific position, as in
  `image(myImage,0,0);`
- To work with the actual pixels, they need to be placed into the `pixels[ ]` array with `loadPixels( )` command.
- To update the screen with the revised pixels, use the `updatePixels( )` command.

You will use all of that information in this lab.

Thinking in 2D – Review In Case You Forgot

Recall from the Assignment 12 that when we work with the pixels[ ] array, we need to use a linear view of the photo because the pixels are listed in row-major-order, that is from the top left corner, going right and then down row-by-row to the lower right. We used a diagram similar to this to describe the situation:



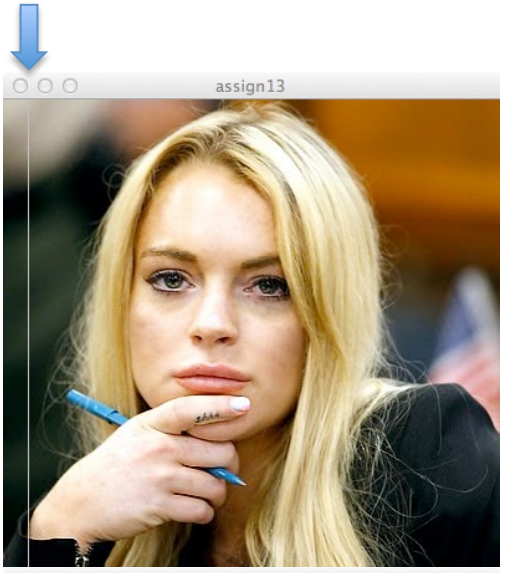
This time, we will write a simple, value returning function that converts from the 2-D view we'd like to use with the photo to the 1-D view that the pixels[ ] array needs.

The function `convert( )` has two integer parameters, the x-position and the y-position. It simply returns the value computed by multiplying `ypos` rows times width, and adding `xpos`. It's a one-line function.

In Court

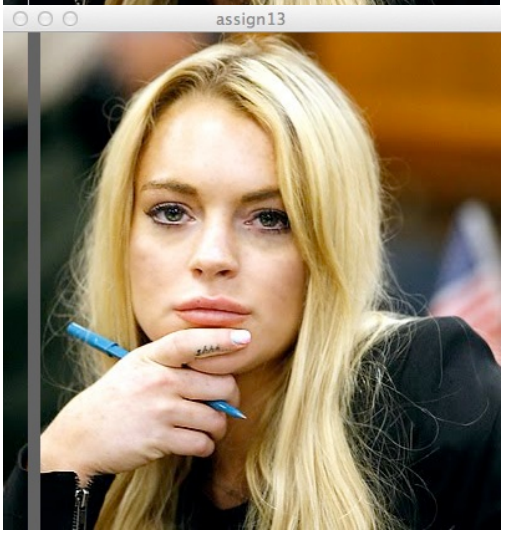
Find a photo of someone who belongs in jail. Place it on the canvas using the steps mentioned in the Reminder above. Placing those operations, except for the last, in `setup( )` is recommended.

Our goal is to draw bars on top of the image.



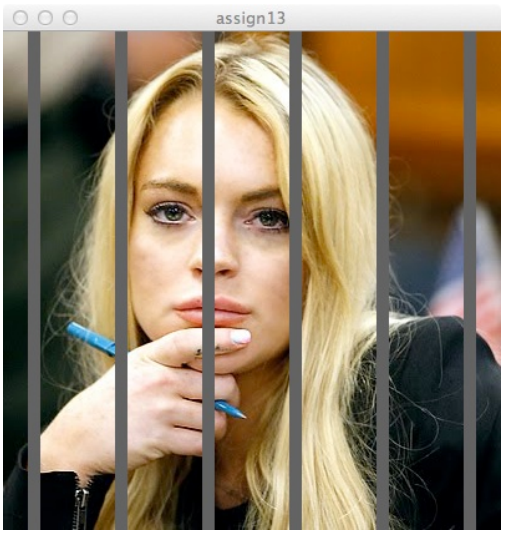
Draw A Line

Working in the `draw( )` function and before the `updatePixels( )` command, we will draw a gray line, one pixel wide, down the left side of the image, say 20 pixels in from the edge. Obviously, to change all of those pixels, we will need a for-loop. To refer to the right pixels in pixels[ ], we can use the `convert(20, i)` function from above, where 20 is the x-position value, and `i`, the y-position value determined by our for-loop, runs from 0 up to `height`. For each of those pixels we want to change it to gray, `color(80)`.



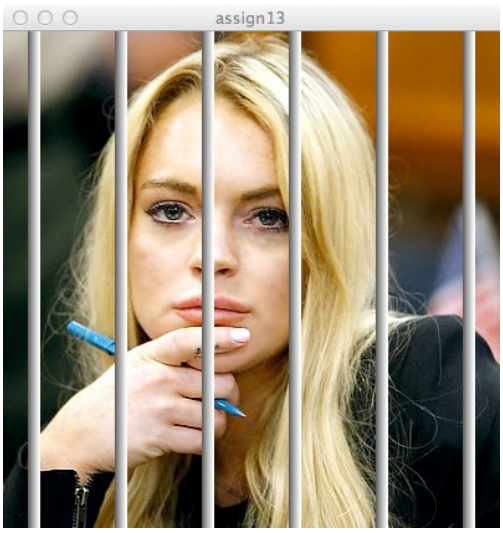
Draw A Thicker Line

The line doesn't look much like a bar, so we need to thicken it. One way to do that is to apply the gray color to pixel columns 20, 21, 22, …, 29, not just column 20. That is, repeat what we did in Draw A Line for 10 pixels rather than one. Doing that only requires that we put the code for Draw A Line *in as the body of another loop*, say on `j`, that goes from 0 up to 10. By adding `j` to 20 in our function call `convert(20+j, i)`.



Draw Lots of Bars

To create a "jail" effect, I'll draw six copies of the bar from the last step, spaced every 70 pixels. To do that, I'll need yet another for-loop, say on `k`, that runs from 0 up to 6, and to put the code from the last step (yes, the loop-within-a-loop) *inside the body of this third loop*. Again, we change our `convert( )` call to include the 70-pixel offset, that is, `convert(20+j+70*k, i)`. Notice that pictures of different sizes will need different numbers of bars, and different spacing.



More Realism

As if being in jail were not real enough, we make the bars look more realistic by making them appear to be more rounded. We do this by adding some highlights, as shown at right.

The problem is that the gray is flat in the earlier pictures. So we lighten it by adding 20 to the gray value for each pixel column making up a bar. Then, the color will get lighter from left to right. Because the width of the bars was handled by the `j` loop, we simply make our color call, `color(80+20*j)`.

Wrap Up

This exercise has introduced the idea of nested loops, a loop within a loop. As we have seen, it is common to create some effect using a loop, and then want to repeat the effect, which requires an additional loop. We accomplish the result by putting one loop in the body of another. As an illustration,

```
for (int i = 0; i < 10; i++) {
   for (int j = 0; j < 10; j++) {
       pixels[convert(100+i, 100+j)] = color(255,0,0);
   }
}
```

which places a red 10 x 10 square in position 100. That is, it does the work that the function `rect(100,100, 10, 10)` would do. The `j`-loop is called the *inner loop*, the `i`-loop is called the *outer loop*.

Turn In

Both partners should place a copy of the commented program and the image in the class dropbox.