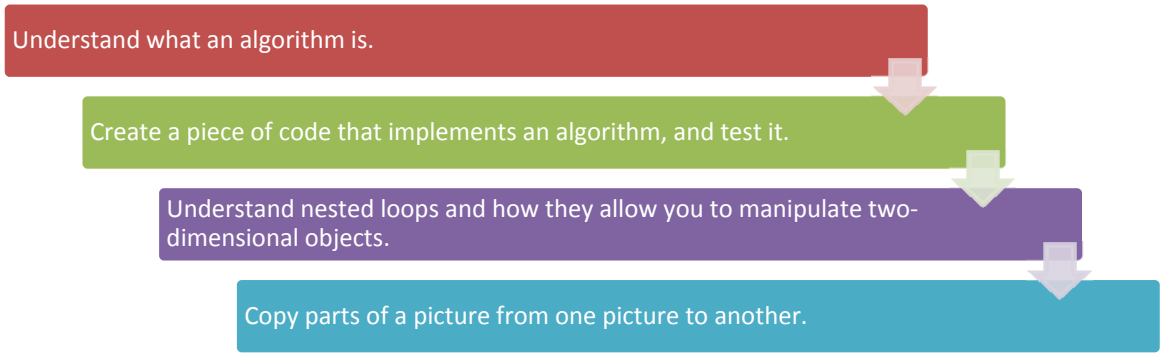
Module III: Summary

Learning Objectives



Programming test concepts

- An *algorithm* is a series of repeatable steps that can be used to solve a particular problem. You can design and describe an algorithm using pseudocode, which you can use to understand the general strategy you want to take. Remember that when you convert this pseudocode to actual code in a language like Java, there are additional details you have to worry about like loss of precision, though these are not critical to the design of the algorithm itself.
- It is often useful to trace through your code and understand what the computer is doing to the values of your variables after each statement is executed. It may be helpful to you to draw boxes to represent graphically what is happening to your variables and the object references and values they contain.
- *Nested loops* are a common tool for dealing with multi-dimensional structures in programming, like the two-dimensional structure we have in images. Nested loops usually look something like this:

```
for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
                <apply transformation (change the color of a pixel, copy it into a specified location, etc)>
        }
}
```

  Nested loops work in exactly the same way as loops normally do, since they are just a loop inside of another loop, but we need to think about when the inner value and the outer value is going to be modified. In any situation, remember that the inner loop will always complete all its iterations for a particular value in the outer loop before the outer loop can continue to the next iteration. (so for example, in the example we go through values like (i=0, j=0), (i=0, j=1), (i=0, j=2)...(i=0, j=width-1), (i=1, j=0), (i=1, j=1), (i=1, j=2), ... etc.
- When writing an algorithm, it often helps to test out your algorithm on small, simple examples first, and then scale up to larger examples. Make sure you test on cases with different characteristics (empty arrays, inputs with even and odd sizes, etc.) to work out potential bugs before you decide that your algorithm works.
- We can use a *comma* (,) to put more than one expression on a single line of code and execute them in order as a single statement. The most common example of this is in a for loop, where we can say for(int i = 0, j = 0; i < 100; i++, j++). In this example, we are initializing two variables, and changing both of them by essentially executing two separate statements in the initialization and the change statement in the for loop. Another use of this is to declare multiple variables of the same type at once. For example at the start of a method you might write int test1, test2;
- A good method should:
    - Do one thing, and do it well
    - Call other methods to do a common task
    - Not repeat something that's in another method. If you find yourself copying code, create a method just for that, and call it from all the other methods that used to have that code
    - Be in the class that contains the data it is working on

Media concepts

- We discussed how to increase or decrease the amount of red (or blue or green) in an image. Remember that we did this by multiplying the red value at each pixel by a fraction, like 0.8 for example. Be aware that in situations like this your result will be a double, but you need to store it back into an int, so you have to *cast* the value back like this value = (int)(value*0.7);
- We can remove all of the red, green, or blue parts of an image by setting their respective values to zero. In this case, we don't care what the current value at the pixel is.
- We learned how to "fake a sunset" by setting the blue and green values each to 0.7 times their original values, which, as you'd expect, makes the image look redder but darker overall.
- We learned how to negate an image by changing each of the RGB values to 255 minus their current value.
- We learned how to make an image greyscale by calculating the overall brightness of a particular pixel and setting the red, green and blue values to the same value determined by that brightness.
- We learned how to solve the problem of mirroring an image across a vertical axis, by thinking about the algorithm on small, simple examples first and then testing it on a larger image.
- To copy a picture to another picture, we can use a set of nested loops like this:

```
for (int srcX = 0, tgtX = 0; srcX < source.getWidth(); srcX++, tgtX++) {
        for (int srcY = 0, tgtY = 0; srcY < source.getHeight(); srcY++, tgtY++) {
                Pixel srcpix = source.getPixel(srcX, srcY);
                Pixel tgtpix = this.getPixel(tgtX, tgtY);
                tgtpix.setColor(srcpix.getColor());
        }
}
```

  This takes advantage of the *comma* operator, which lets you execute multiple operations you might normally put in separate statements (separated by semicolons) as one statement. There are other ways to use this operator, but because of the fact the for loop requires exactly three statements in a particular order, the comma is especially useful here.
- Use picture.explore() to find the coordinates of a specific part of a picture, or to test your code by checking the RGB values for a specific pixel.
- Set up your media path so that you can refer to pictures by file name without typing in the whole path.
- There are many simple transformations that can be done on pictures. The last module changing the colors of pixels, for example. This module covered the following transformations:
  - A picture can be mirrored around an axis.
  - A picture (or part of a picture) can be copied into another picture.
  - A picture can be scaled down in size by grabbing one pixel (and copying it into a new, half-sized image), skipping the next, and so on and so forth. In doing this, it is important to make sure that we don't throw out the last row and column of images that have an odd number in their width and/or height. **(skipped)**
  - A picture can be scaled up by copying each pixel twice, into two neighboring slots in the target image. **(skipped)**
  - A picture can be rotated left or right. **(skipped)**