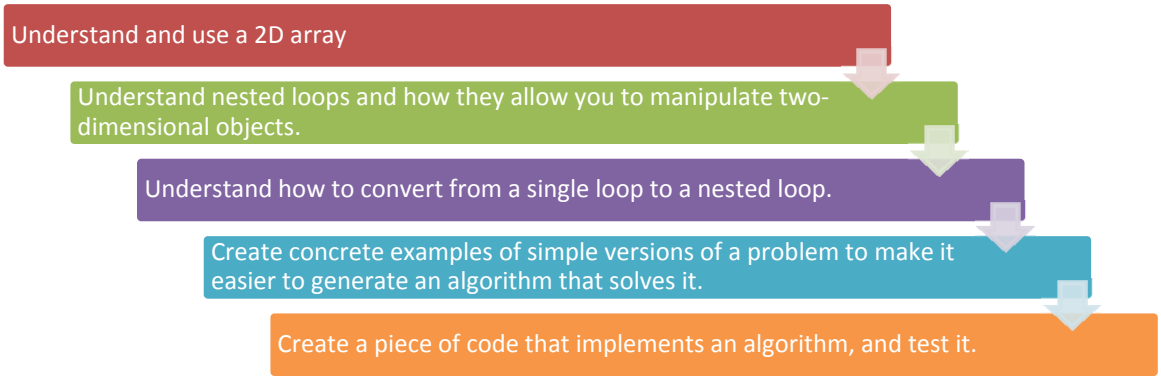
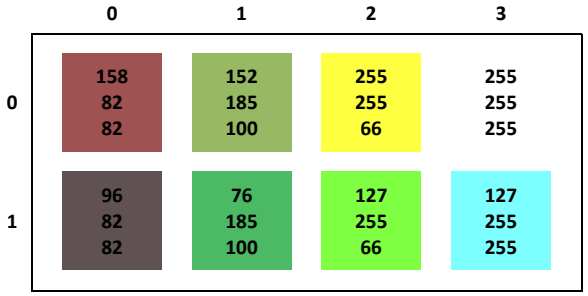
Module III: Nested Loops

Learning Objectives



Review: Pictures and Two-Dimensional Arrays

As we explored earlier, a picture can be thought as a two-dimensional array of pixels (seen in module 2). If we think of a picture this way, each pixel has an x value (its horizontal location) and a y value (its vertical location). We learned that the syntax pictureObj.getPixel(**x,y**) can be used to retrieve the pixel at a specific location. This image illustrates the concept:



This 2D array represents a picture that is very tiny. It is 4 pixels wide and 2 pixels high. getPixel(0,0) would return the rust colored pixel, whose RGB values are (198, 82, 82). getPixel(3,1) would return the grey pixel whose RGB values are (217, 217, 217)

Nested Loops

You already know how to use loops, and you should have some experience with accessing and manipulating arrays using while and for loops. But how do we use loops with two-dimensional arrays?

Let's start by thinking about an algorithm for retrieving pixels in an orderly fashion. We could go left to right, and top to bottom, as follows:

- Get all of the pixels in the first row. Assuming a 640x480 image, this would result in getting the pixel at *(0, 0)*, then *(1, 0)*, then *(2, 0)* and so on up to *(639, 0)*.
- Then get all of the pixels in the second row. This would result in getting the pixel at *(0, 1)*, then *(1, 1)*, then *(2, 1)* and so on up to *(639, 1)*
- ...
- Then get all of the pixels in the last row. This would result in getting the pixel at *(0, 479)*, then *(1, 479)*, then *(2, 479)* and so on up to *(639, 479)*

But how is this algorithm implemented? The most common solution to this problem is to use *nested loops*, which is when we have a single loop on the outside to loop through the rows (top to bottom) and another one *inside* that loop to loop through the columns (left to right). We can refine the pseudo code given above as follows:

```
for each row r (0 to height-1):
        for each column c (0 to width-1):
                do something with the pixel at (c, r)
```

nested loop

(definition) A loop that is placed inside another loop. This will cause the inner loop to iterate across a set of values once for each iteration of the outer loop.

When one loop is nested *within* another, the inner loop is executed, *in full* each time the outer loop goes through one iteration. In the example above, the outer loop's first iteration is concerned with the first (r=0) row. The inner loop will iterate through *every column* (0-width-1) of the 0th row. Then the outer loop's first iteration will complete, and it will begin the second iteration, which is concerned with the second row (r=1). Again, the inner loop will iterate through every column in that row. And so on, until the final (r=height-1) row is completed.

Observation... It doesn't actually matter whether you iterate through a picture row first or column first. Either way, you will eventually get to every pixel in the image exactly once. Let's try it.

Here is a simple template with nested loops that you might use to go through the pixels in an image.

```
Pixel pixelObj;
for (int y = 0; y < this.getHeight(); y++) {   // loop through the rows
   for (int x = 0; x < this.getWidth(); x++) {    // loop through the columns
     //get the current pixel
     pixelObj = this.getPixel(x, y);

     // do something with the pixel
     // such as pixelObj.setColor(...)
   }
}
```

Note... Do you see how we indent some lines more than others? This helps with the readability of the code. Every time you write a line of code that ends in a {, you should start the next line more indented. When you write the close }, you should stop indenting. This helps to visually indicate where blocks of code start and end, and will help you to see what the scope of your variables is as well.

It can be tricky to convert between rows and columns and the x-y coordinates of a pixel. Really that the x coordinates increase as we go from left to right (along the x-axis) and the y coordinates increase as we go from top to bottom (along the y-axis). Thus, we can label the x-y coordinates starting with (0,0) at the top left and ending with (w,h) at the bottom right. This is different from the coordinate system you may be used to, where the y-axis increases from bottom to top. Even more confusing, the x coordinate represents the *column number* of the current pixel, while the y coordinate represents the *row number* of the current pixel. This may seem non-intuitive at first, so try drawing this on paper to see how it works.

Lightening an Image

Now that we've explored the pseudocode and looked over a template for how to implement nested loops, it's time to try using them. We'll write a loop to lighten the color of an image. To lighten the color of an image, we can call pixel.brighter() for each pixel in the image. Let's start with some pseudocode. Our algorithm should look something like this:

```
for each row r (0 to height-1):
        for each column c (0 to width-1):
                Get the pixel at (r, c)
                Lighten the color of that pixel
                Set the color of that pixel to the lighter color
```

Here is an implementation of this algorithm:

```
public void lighten() {
   Pixel pixel;
   Color c;

   for (int y = 0; y < this.getHeight(); y++) {  //loop through the rows
      for (int x = 0; x < this.getWidth(); x++) {            //loop through the columns
         pixel = this.getPixel(x, y);
         //make a color that is brighter than the current color of the pixel
         c = pixel.getColor().brighter();
         //set the color of the pixel to the lighter color
         pixel.setColor(c);
      }
   }
}
```

If you want to try it, check out the example MediaSix and add the following:

```
String file = "<your file path>\\MediaSix\\src\\mediasix\\sunset.jpg";
Picture pictureObj = new Picture(file);
pictureObj.lighten();
pictureObj.repaint();
```

Here is the result of running it:

On the left is the original image. On the right is a lighter version of the image.

learn by doing

Create the method clearBlue( ) to use a nested loop to loop through all the pixels setting blue to 0. Run the method to check that it works.

solution

```
public void clearBlue() {
      Pixel pixel;
      Color c;

      for (int y = 0; y < this.getHeight(); y++) {  //loop through the rows
         for (int x = 0; x < this.getWidth(); x++) {  //loop through the columns
            pixel = this.getPixel(x,y);
            //set the blue on the pixel to 0
            pixel.setBlue(0);
         }
      }
}
```

Vertical Mirroring

Suppose we want to mirror an image vertically. Pretend you could place a mirror in the middle of an image, so that you would see the left side of the image mirrored in the right.

An image (left) and the same image mirrored along a vertical line down the center (right).

How was this done? We'll need to create an algorithm for mirroring images before we can implement anything. The best way to figure out the correct algorithm is to work through a really simple example. Let's try to solve the problem for a really small picture that has only 15 pixels (5x3). We can also simplify the problem by imagining a two-dimensional array full of numbers instead of a picture.

Example

Consider the following two-dimensional 5x3 array. The first row/column shows the index values for the array (0-4 across and 0-2 down). The array itself contains the numbers 1,2,3,4,5 in the first and last row, and 5,4,3,2,1 in the middle row.

	0	1	2	3	4
0	1	2	3	4	5
1	5	4	3	2	1
2	1	2	3	4	5

Let's see what that array would look like if we created a vertical mirror around the middle column, which can be obtained by getting the midpoint of each array. To make it easier to see, the numbers that are new are in italics and the numbers in the mirror column are shown in bold.

	0	1	2	3	4
0	1	2	***3***	*2*	*1*
1	5	4	***3***	*4*	*5*
2	1	2	***3***	*2*	*1*

Can you figure out the algorithm for this? It should be easier than thinking about a more complex image.

If that is too complex, try to solve the problem on an even smaller array. In fact, we'll use one so small that it's just a single row of an array.

	0	1	2	3	4	5	6
0	5	4	4	5	3	3	1

learn by doing

Draw the row shown above on paper. Now draw a version of it that is mirrored along the central vertical axis. Assume that left side will have dominance.

Does the resulting row have a 3 in it?

Solution:

The mirror axis is column 3 in this array (so everything to the left of column 3 remains unchanged).

Think about how the value of column 4 changes after it is mirrored. From which column is column 4 getting its value? How about columns 5 and 6?

Therefore, the mirror axis is column 3 in this array (so everything to the left of column 3 remains unchanged). The new row will have the same values columns 4 and 5 as the current one has in columns 1 and 2. From left to right, it will have: 5, 4, 4, 5, 4, 4, 5. Answer is NO.

learn by doing

Try to write down an algorithm for mirroring a row of an array. It should contain step-by-step instructions. If you follow them without deviating, can you successfully mirror all of the rows in the examples we just went through? That's a sign that you may have something that could be part of a successful mirroring algorithm.

Hint …

- Since mirroring is done around the central column of an array, you may need to calculate the index of the middle column.
- You can mirror a row by copying an entry to the right of the central column to a spot on the left of the central column.
- Can you think of a way to describe where you copy an entry to in terms of the row's width?
- The 0th element ends up in at width-0 ; the 1st element ends up at width-1, the 2nd element ends up at width - 2, and so on.

Solution

Even if you put down a failed attempt, it's a first step toward becoming a programmer!

Here is an algorithm:

Loop through each entry in the row, but only go as far as the midpoint (from 0 to x < width/2)

.... [inside the loop] Set the value of the item at right of midpoint (width - 1 - x, y) to the same value as the item at left of midpoint (x, y)

Even though this only solves part of the bigger problem (mirroring a 2d array or image consisting of multiple rows), it is a very important first step. Programmers know that breaking a problem down into manageable pieces that can be added back together is an excellent way to solve programming problems.

Now that you've figured out how to mirror a row, see if you can mirror this entire array

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

learn by doing

Draw the array shown above on paper. Now draw a version of it that is mirrored along the central vertical axis.

Does the resulting array have a 6 in it?

Solution

YES. Start by just mirroring the first row about its middle element. Now fill in the rest of the array. The new array has no 6 or 3 or 9 in it.

Do you have an idea for an algorithm yet? If you don't try mirroring a couple more arrays until you do. It is important to try this yourself. Figuring out algorithms to solve problems within the constraints of the computer system/language they are using is one of the most common (and most interesting!) things that programmers do.

learn by doing

Try to write down an algorithm for mirroring an array. It should contain step-by-step instructions. If you follow them without deviating, can you successfully mirror all of the arrays in the examples we just went through? That's a sign that you may have a successful algorithm.

Hint…

- Start by figuring out how you will mirror each row of pixels.
- You will need to loop through the row and copy pixels from the left of the midpoint to the right of the midpoint.
- The pixel at location x should be copied to position width-1-x.
- You only need to loop through half the row, since at that point you will have filled the other half in.
- You will need an outer loop that loops through all of the rows.

Solution

Even if you put down a failed attempt, it's a first step toward becoming a programmer!

Here is an algorithm:

For each row in the pixel array (from 0 to height-1)

.... Loop through columns, but only go to the midpoint (from 0 to width/2)

..... Set the value of the item at right of the midpoint (width - 1 - x, y) to the same value as the item at left of the midpoint (x, y).

Now that we have some pseudocode, it's time to translating that into an implementation. There are several decisions that need to be made:

- One thing we know we'll need is the midpoint of the array. We need midpoint to construct the 'middle column' that serves as a mirror for the array. We can get this using int midpoint = this.getWidth()/2;
- We also will need to decide what type of loop to use. Since we want to loop through all of the rows, but only half of the pixels in each row, we will need to use nested loops here. It will look something like:

```
// loop through the rows (y values)
for (int y = 0; y < this.getHeight(); y++) {
  //loop through the columns (x values)
  for (int x = 0; x < midpoint; x++)   {
    ...
  }
}
```

- Finally, we will need a way to decide what pixels to update in each iteration. This is a core piece of the algorithm and was specified in the pseudocode. We will need to decide what it means for pictures, though. What we want to do is copy the *color* of each pixel to the left of the mirror line into the pixel to the right of the mirror line. We will probably need a variable to hold the pixel on the left of the mirror axis (this pixel will be copied), and a variable to hold the pixel that is on the right of the mirror axis (this pixel will be modified). It should look something like:

```
Pixel leftPixel = this.getPixel(x, y);
Pixel rightPixel = this.getPixel(this.getWidth() - 1 - x, y);
rightPixel.setColor(leftPixel.getColor());
```

Here's the final, working implementation of our algorithm:

```
/**
 * mirrorVertical mirrors this image around a vertical axis through its midpoint
```

```
 * It works by copying the color of each pixel on the left side of the
 * mirror axis into the corresponding pixel on the right side of the mirror axis
 */
public void mirrorVertical() {
  Pixel left, right;

  for (int y = 0; y < this.getHeight(); y++) {     // loop through the rows
    for (int x = 0; x < this.getWidth() / 2; x++) {    // loop from 0 to just before the mirror point
      left = this.getPixel(x, y);      // get the pixel that we are copying from
      right = this.getPixel(this.getWidth() - 1 - x, y);
      right.setColor(left.getColor());
    }
  }
}
```

Go ahead and try mirrorVertical().

Walkthrough

At the beginning of the method, we declare variables left and right to store the Pixel on each side of the midpoint column. There are two loops that are nested: the outer loop iterates through the rows of the array, and the inner loop iterates through the columns of the array. That is, for each row, a full iteration of the array by column is performed from the beginning to the end. In this case, the outer loop iterates over all the rows, while the inner loop iterates over only half of the columns. This is because we are mirroring the image vertically along the midpoint column: we do not need to traverse through all columns to do this.

For each row, for each column (up until the midpoint), we get the Pixel at the current x and y position and store it into the variable left. Then, we get the pixel that is opposite to the left pixel and stores it into the variable right. We retrieve the color of the left pixel and set the color of the right pixel to match that color. We repeat this process for each row from the column at index 0 to the column just before the midpoint column.

The outer loop exits when we have completed the mirroring of the last row. At that point, every row (and thus the whole image) has been mirrored vertically.

Horizontal Mirroring

Now that you have seen how vertical mirroring works, you should be able to see how we could create a similar algorithm to mirror an image horizontally. In this case, we'll mirror around a horizontal line through the center of the image, like a reflection in a lake.

As before, we'll need to create an algorithm for this problem before we can implement a solution. Let's start by reconsidering an array we tested our ideas on earlier.

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

learn by doing

Implement a working version of mirrorHorizontal(). Test it out!

Hint …

- Start with mirrorVertical and copy it. Now think about what should stay the same and what needs to be modified.
- You may want to swap the inner and outer loop, so that you loop through the columns (outer loop) and rows (inner loop).
- You will need to mirror around the midpoint of the column (height, or y value).

Solution

You will need to copy pixels from above that midpoint to below it. You will need to calculate the position to copy to by subtracting the current y position from the image height (- 1). The final algorithm should look like this:

```
/**
   * mirrorHorizontal mirrors this image around a horizontal axis through its
   * midpoint It works by copying the color of each pixel on the top of
   * the mirror axis into the corresponding pixel on the bottom of the
   * mirror axis
   */
  public void mirrorHorizontal() {
    Pixel top, bottom;

    for (int x = 0; x < this.getWidth(); x++) {       // loop through the cols
      for (int y = 0; y < this.getHeight() / 2; y++) {          // loop from 0 to just before the mirror point
        top = this.getPixel(x, y);              // get the pixel that we are copying from
        bottom = this.getPixel(x, this.getHeight() - 1 - y);
        bottom.setColor(top.getColor());
      }
    }
  }
```

learn by doing

Implement a working version of mirrorHorizontalReverse() that copies the bottom half of the image to the top instead of the top half to the bottom. Test it out and paste the code in below. Using the beach image instead of the sunset image.

Solution

The main difference between this and mirrorHorizontal is where you copy pixels from, and where you copy them to. You will want to copy pixels from below the mirror axis to above it. The result should look something like this:

```
public void mirrorHorizontalReverse() {
    Pixel top, bottom;

    for (int x = 0; x < this.getWidth(); x++) {       // loop through the cols
      for (int y = 0; y < this.getHeight() / 2; y++) {          // loop from 0 to just before the mirror point
        top = this.getPixel(x, y);              // get the pixel that we are copying from
        bottom = this.getPixel(x, this.getHeight() - 1 - y);
        top.setColor(bottom.getColor());
      }
    }
  }
```

Summary

In summary, we have revisited the topic of two-dimensional arrays, which have columns and rows that can be indexed separately to specify any x,y location. In images, this is done with getPixel(**x,y**).

Next, we introduced nested loops, and showed how they can be used to work with two-dimensional arrays. One loop is used for the x direction and the other loop for the y direction. When they are nested, the inner loop will do a complete set of iterations once for each iteration of the outer loop. In a sense the inner loop is running very quickly over and over again, while the outer loop is waiting patiently for the inner loop at each step.

We then focused on trying to solve the problem of mirroring an image around an axis. We developed our own algorithm for this by creating several small versions of a problem and solving them. Along the way we realized what had to be done differently in rows versus columns. We used these examples to develop a more general algorithm which we then translated into code.

This is a process you'll be likely to repeat over and over again.

While learning about algorithms, we developed an approach to mirror pictures. To do so, we select a mirror axis (midpoint of a row or column). We then copy pictures from one side of the row or column to the other side

Note: All the code implementation is available in the Netbeans project MediaSix