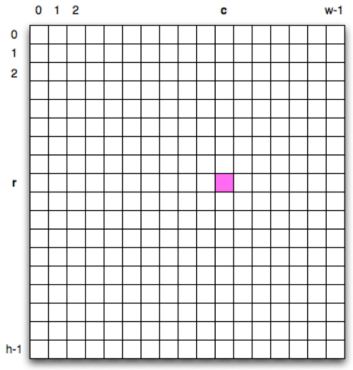
7.8. Nested Iteration: Image Processing

Two dimensional tables have both rows and columns. You have probably seen many tables like this if you have used a spreadsheet program. Another object that is organized in rows and columns is a digital image. In this section we will explore how iteration allows us to manipulate these images.

A **digital image** is a finite collection of small, discrete picture elements called **pixels**. These pixels are organized in a two-dimensional grid. Each pixel represents the smallest amount of picture information that is available. Sometimes these pixels appear as small "dots".

Each image (grid of pixels) has its own width and its own height. The width is the number of columns and the height is the number of rows. We can name the pixels in the grid by using the column number and row number. However, it is very important to remember that computer scientists like to start counting with 0! This means that if there are 20 rows, they will be named 0,1,2, and so on through 19. This will be very useful later when we iterate using range.

In the figure below, the pixel of interest is found at column **c** and row **r**.



7.8.1. The RGB Color Model

Each pixel of the image will represent a single color. The specific color depends on a formula that mixes various amounts of three basic colors: red, green, and blue. This technique for creating color is known as the **RGB Color Model**. The amount of each color, sometimes called the **intensity** of the color, allows us to have very fine control over the resulting color.

The minimum intensity value for a basic color is 0. For example if the red intensity is 0, then there is no red in the pixel. The maximum intensity is 255. This means that there are actually 256 different amounts of intensity for each basic color. Since there are three basic colors, that means that you can create 256^3 distinct colors using the RGB Color Model.

Here are the red, green and blue intensities for some common colors. Note that "Black" is represented by a pixel having no basic color. On the other hand, "White" has maximum values for all three basic color components.

Color	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
White	255	255	255
Black	0	0	0
Yellow	255	255	0
Magenta	255	0	255

In order to manipulate an image, we need to be able to access individual pixels. This capability is provided by a module called **image**, provided in ActiveCode [1]. The image module defines two classes: `Image` and `Pixel`.

[1] If you want to explore image processing on your own outside of the browser you can install the cImage module from http://pypi.org.

Each Pixel object has three attributes: the red intensity, the green intensity, and the blue intensity. A pixel provides three methods that allow us to ask for the intensity values. They are called `getRed`, `getGreen`, and `getBlue`. In addition, we can ask a pixel to change an intensity value using its `setRed`, `setGreen`, and `setBlue` methods.

Method Name	Example	Explanation
Pixel(r,g,b)	Pixel(20,100,50)	Create a new pixel with 20 red, 100 green, and 50 blue.
getRed()	r = p.getRed()	Return the red component intensity.
getGreen()	r = p.getGreen()	Return the green component intensity.
getBlue()	r = p.getBlue()	Return the blue component intensity.
setRed()	p.setRed(100)	Set the red component intensity to 100.
setGreen()	p.setGreen(45)	Set the green component intensity to 45.

setBlue()	p.setBlue(156)	Set the blue component intensity to 156.

In the example below, we first create a pixel with 45 units of red, 76 units of green, and 200 units of blue. We then print the current amount of red, change the amount of red, and finally, set the amount of blue to be the same as the current amount of green.

Save & Run 4/30/2021, 9:18:53 PM - 2 of 2

```
import image

p = image.Pixel(45, 76, 200)
print(p.getRed())
p.setRed(66)
print(p.getRed())
p.setBlue(p.getGreen())
print(p.getGreen(), p.getBlue())

```

```
45
66
76 76
```

Activity: 1 -- ActiveCode (ac14_7_1)

Check your understanding

moreiter-7-1: If you have a pixel whose RGB value is (50, 0, 0), what color will this pixel appear to be?

- A. Dark red
- B. Light red
- C. Dark green
- D. Light green

Check me Compare me

✔ Because all three values are close to 0, the color will be dark. But because the red value is higher than the other two, the color will appear red.

Activity: 2 -- Multiple Choice (question14_7_1)

7.8.2. Image Objects

To access the pixels in a real image, we need to first create an `Image` object. Image objects can be created in two ways. First, an Image object can be made from the files that store digital images. The image object has an attribute corresponding to the width, the height, and the collection of pixels in the image.

It is also possible to create an Image object that is "empty". An `EmptyImage` has a width and a height. However, the pixel collection consists of only "White" pixels.

We can ask an image object to return its size using the `getWidth` and `getHeight` methods. We can also get a pixel from a particular location in the image using `getPixel` and change the pixel at a particular location using `setPixel`.

The Image class is shown below. Note that the first two entries show how to create image objects. The parameters are different depending on whether you are using an image file or creating an empty image.

Method Name	Example	Explanation
Image(filename)	img = image.Image("cy.png")	Create an Image object from the file cy.png.
EmptyImage()	img = image.EmptyImage(100,200)	Create an Image object that has all "White" pixels
getWidth()	w = img.getWidth()	Return the width of the image in pixels.
getHeight()	h = img.getHeight()	Return the height of the image in pixels.
getPixel(col,row)	p = img.getPixel(35,86)	Return the pixel at column 35, row 86.
setPixel(col,row,p)	img.setPixel(100,50,mp)	Set the pixel at column 100, row 50 to be mp.

Consider the image shown below. Assume that the image is stored in a file called "luther.jpg". Line 2 opens the file and uses the contents to create an image object that is referred to by `img`. Once we have an image object, we can use the methods described above to access information about the image or to get a specific pixel and check on its basic color intensities.

Save & Run 4/30/2021, 9:20:23 PM - 2 of 2

```
import image
img = image.Image("luther.jpg")

print(img.getWidth())
print(img.getHeight())

p = img.getPixel(45, 55)
print(p.getRed(), p.getGreen(), p.getBlue())

```

```
400
244
165 161 158
```

Activity: 3 -- ActiveCode (ac14_7_2)

When you run the program you can see that the image has a width of 400 pixels and a height of 244 pixels. Also, the pixel at column 45, row 55, has RGB values of 165, 161, and 158. Try a few other pixel locations by changing the `getPixel` arguments and rerunning the program.

Check your understanding

moreiter-7-2: Using the previous ActiveCode example, select the answer that is closest to the RGB values of the pixel at row 100, column 30? The values may be off by one or two due to differences in browsers.

- A. 149 132 122
- B. 183 179 170
- C. 165 161 158
- D. 201 104 115

Check me Compare me

✔ Yes, the RGB values are 183 179 170 at row 100 and column 30.

Activity: 4 -- Multiple Choice (question14_7_2)

7.8.3. Image Processing and Nested Iteration

Image processing refers to the ability to manipulate the individual pixels in a digital image. In order to process all of the pixels, we need to be able to systematically visit all of the rows and columns in the image. The best way to do this is to use **nested iteration**.

Nested iteration simply means that we will place one iteration construct inside of another. We will call these two iterations the **outer iteration** and the **inner iteration**. To see how this works, consider the iteration below.

```
for i in range(5):
    print(i)
```

We have seen this enough times to know that the value of `i` will be 0, then 1, then 2, and so on up to 4. The `print` will be performed once for each pass. However, the body of the loop can contain any statements including another iteration (another `for` statement). For example,

```
for i in range(5):
    for j in range(3):
        print(i, j)
```

The `for i` iteration is the outer iteration and the `for j` iteration is the inner iteration. Each pass through the outer iteration will result in the complete processing of the inner iteration from beginning to end. This means that the output from this nested iteration will show that for each value of `i`, all values of `j` will occur.

Here is the same example in activecode. Try it. Note that the value of `i` stays the same while the value of `j` changes. The inner iteration, in effect, is moving faster than the outer iteration.

Save & Run 4/30/2021, 9:20:48 PM - 2 of 2 Show in CodeLens

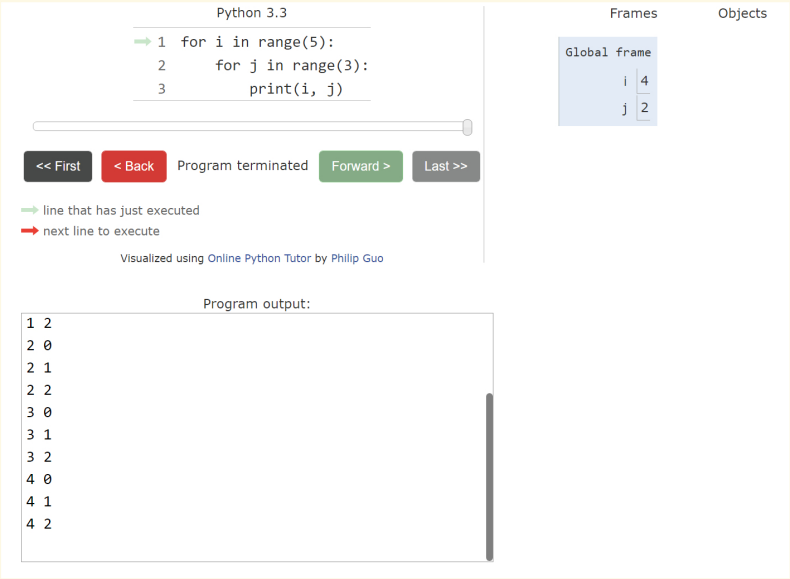
```
for i in range(5):
    for j in range(3):
        print(i, j)

```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
3 0
3 1
3 2
4 0
4 1
4 2
```

Activity: 5 -- ActiveCode (ac14_7_3)

Another way to see this in more detail is to examine the behavior with codelens. Step through the iterations to see the flow of control as it occurs with the nested iteration. Again, for every value of `i`, all of the values of `j` will occur. You can see that the inner iteration completes before going on to the next pass of the outer iteration.



Activity: 6 -- CodeLens: (clens14_7_1)

Our goal with image processing is to visit each pixel. We will use an iteration to process each row. Within that iteration, we will use a nested iteration to process each column. The result is a nested iteration, similar to the one seen above, where the outer `for` loop processes the rows, from 0 up to but not including the height of the image. The inner `for` loop will process each column of a row, again from 0 up to but not including the width of the image.

The resulting code will look like the following. We are now free to do anything we wish to each pixel in the image.

```
for row in range(img.getHeight()):
    for col in range(img.getWidth()):
        # do something with the pixel at position (col,row)
```

One of the easiest image processing algorithms will create what is known as a **negative** image. A negative image simply means that each pixel will be the opposite of what it was originally. But what does opposite mean?

In the RGB color model, we can consider the opposite of the red component as the difference between the original red and 255. For example, if the original red component was 50, then the opposite, or negative red value would be `255-50` or 205. In other words, pixels with a lot of red will have negatives with little red and pixels with little red will have negatives with a lot. We do the same for the blue and green as well.

The program below implements this algorithm using the previous image (luther.jpg). Run it to see the resulting negative image. Note that there is a lot of processing taking place and this may take a few seconds to complete. In addition, here are two other images that you can use (cy.png and goldygopher.png).

cy.png

goldygopher.png

Change the name of the file in the `image.Image()` call to see how these images look as negatives. Also, note that there is an `exitonclick` method call at the very end which will close the window when you click on it. This will allow you to "clear the screen" before drawing the next negative.

Save & Run 4/30/2021, 9:22:29 PM - 2 of 2

```
1  import image
2
3  img = image.Image("luther.jpg")
4  win = image.ImageWin(img.getWidth(), img.getHeight())
5  img.draw(win)
6  img.setDelay(1,15)   # setDelay(0) turns off animation
7
8  for row in range(img.getHeight()):
9      for col in range(img.getWidth()):
10         p = img.getPixel(col, row)
11
12         newred = 255 - p.getRed()
13         newgreen = 255 - p.getGreen()
14         newblue = 255 - p.getBlue()
15
```

Activity: 7 -- ActiveCode (ac14_7_4)

Let's take a closer look at the code. After importing the image module, we create an image object called `img` that represents a typical digital photo. We will update each pixel in this image from top to bottom, left to right, which you should be able to observe. You can change the values in `setDelay` to make the program progress faster or slower.

Lines 8 and 9 create the nested iteration that we discussed earlier. This allows us to process each pixel in the image. Line 10 gets an individual pixel.

Lines 12-14 create the negative intensity values by extracting the original intensity from the pixel and subtracting it from 255. Once we have the `newred`, `newgreen`, and `newblue` values, we can create a new pixel (Line 15).

Finally, we need to replace the old pixel with the new pixel in our image. It is important to put the new pixel into the same location as the original pixel that it came from in the digital photo.

Try to change the program above so that the outer loop iterates over the columns and the inner loop iterates over the rows. We still create a negative image, but you can see that the pixels update in a very different order.

Other pixel manipulation

There are a number of different image processing algorithms that follow the same pattern as shown above. Namely, take the original pixel, extract the red, green, and blue intensities, and then create a new pixel from them. The new pixel is inserted into an empty image at the same location as the original.

For example, you can create a **gray scale** pixel by averaging the red, green and blue intensities and then using that value for all intensities.

From the gray scale you can create **black white** by setting a threshold and selecting to either insert a white pixel for a black pixel into the empty image.

You can also do some complex arithmetic and create interesting effects, such as Sepia Tone

Check your understanding

moreiter-7-3: What will the following nested for-loop print? (Note, if you are having trouble with this question, review CodeLens 3).

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

```
a.

0 0
0 1
1 0
```

```
1 1
2 0
2 1

b.

0   0
1   0
2   0
0   1
1   1
2   1

c.

0   0
0   1
0   2
1   0
1   1
1   2

d.

0   1
0   1
0   1
```

- (•) A. Output a
- ( ) B. Output b
- ( ) C. Output c
- ( ) D. Output d

Check me | Compare me

✔ i will start with a value of 0 and then j will iterate from 0 to 1. Next, i will be 1 and j will iterate from 0 to 1. Finally, i will be 2 and j will iterate from 0 to 1.

Activity: 8 -- Multiple Choice (question14_7_3)

moreiter-7-4: What would the image produced from ActiveCode box 16 look like if you replaced the lines:

```
newred = 255 - p.getRed()
newgreen = 255 - p.getGreen()
newblue = 255 - p.getBlue()
```

with the lines:

```
newred = p.getRed()
newgreen = 0
newblue = 0
```

- (•) A. It would look like a red-washed version of the bell image
- ( ) B. It would be a solid red rectangle the same size as the original image
- ( ) C. It would look the same as the original image
- ( ) D. It would look the same as the negative image in the example code

Check me | Compare me

✔ Because we are removing the green and the blue values, but keeping the variation of the red the same, you will get the same image, but it will look like it has been bathed in red.

Activity: 9 -- Multiple Choice (question14_7_4)

You have attempted 10 of 9 activities on this page

7.7. Traversal and the for Loop: By Index">
versal and the for Loop: By Index">
7.9. Printing Intermediate Results">
✔ Completed. Well Done!

| Back to top