Lab #4

Working with Ranges of Pixels

Introduction

In this lab you will use the following image(available on the course web site): beach.jpg

Activity A: Understanding how range() works

We start with an introduction to the range() function that is part of python/jython. This is a function that returns a list of numbers that the computer can consider one at a time. You will see this a lot over the rest of the semester so it's essential that you understand what is going on.

[Q1] For each of the following uses of range, first PREDICT what will be returned when you type this into the interactions area. Once you have made a prediction you should test your results. That is, test the command out and see if you were correct.

Command	Prediction	Actual result
print range(0,5)		
print range(3,10)		
print range(9,3)		
print range(2,10,1)		
print range(2,10,2)		
print range(2,10,3)		
print range(10,2,2)		
print range(10,2,-2)		

[Q2] Write a general statement about how the two parameter version of range works. For example, if we write "print range(x,y)" where x and y are two integers where x<y, what numbers get listed.

[Q3] Write a general statement about how the two parameter version of range works. For example, if we write "print range(x,y)" where x and y are two integers where x>y, what numbers get listed.

[Q4] Write a general statement about how the three parameter version of range works. For example, if we write "print range(a,b,c)" where a and b are two integers where a<b, what numbers get listed.

Now that you know how range works, let's look at what happens when you use range as part of a loop. In the Program Area of JES type in the following small function and save the space as "lab5.py"

```
def demo():
  for x in range(0,5):
    print x
```

Then run this demo:

```
>>> demo()
```

[Q5] What is printed when you run demo()?

But consider what happens when we put one for loop inside of another for loop. Copy the following code into the program area and run it.

```
def demo2():
    for x in range(0,3):
      for y in range(5,9):
          print str(x)+","+str(y)
```

[Q6] What is printed when you run demo2()?

Look at this result carefully. Notice that when the program first sees the "for x" loop it sets x to its first value (x=0). Then the computer sees the "for y" loop and sets y to its first value (y=5). At this point the computer sees the print statement and prints the two values of x and y - we print "0 5"

Notice then that when the computer comes back around to run the next part of the loop (the **for** statement) that it goes back around and runs the next value for y. In other words, it stays with the *inner* loop. The next value of y is 6. Therefore, the computer prints "0 6" The next value of y is 7 so the computer prints "0 7". Finally, the last value of y is 8, so the computer prints "0 8"

At this point, the y loop has completed so the program can't run it anymore. Since the inner loop has completed the computer goes back around to the outer loop – the "for x" loop. The computer sets the next value of x which is 1. Now, when the computer comes back down to the "for y in range(5,8)" loop it treats it as though it hasn't seen it before – in other words it starts that loop all over again, setting y to 5. Thus, the next print statement is "1 5"

At this point the y loop runs through its entire process again so we see "1 5", "1 6" , "1 7", and "1 8".

Again, when the y loop is finished the computer goes back up to the x loop to get its next value (x=2) and then starts the y loop all over again.

The outer loop, the x loop, runs three times. The inner loop – the y loop – runs four times EACH time it runs. Since it runs three times due to the x loop, there are a total of 12 print statements.

Most of the code that we write for pictures can use a set of nested loops like this.

Consider the following program from last week:

```
def increaseGreen(picture):
  for pixel in getPixels(picture):
    greenValue = getGreen(pixel)
    setGreen(pixel, greenValue*1.5)
```

Recall that this program looked at each pixel in the entire picture, one pixel at a time, and changed them to have a higher value of green. You will even recall that we put a repaint() inside of there that allowed us to see this happen.

Last week we wrote it this way because we didn't actually care WHERE the pixels were in the picture. We just wanted them all to change. But now we are going to start working with programs where we DO care where pixels are in the picture. To do this we move from a single loop solution as was in the code above, to a nested, double loop solution.

Look at the following function:

```
def increaseGreen2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      pixel = getPixel(picture,x,y)
      greenValue = getGreen(pixel)
      setGreen(pixel, greenValue*1.5)
```

If you think through what you just observed in demo2() then you know that this is going to start by setting x to be equal to zero. It then moves to the for y loop and sets y to zero. It then goes in and retrieves the pixel at (0,0) and stores it in the variable named pixel. At this point you are exactly where you were last week and you can set that pixel to a brighter green. We continue in this manner getting the next value of y and the NEXT value of y until we have changed every y value in the entire column where x=0. Then we move to x=1 and start over.

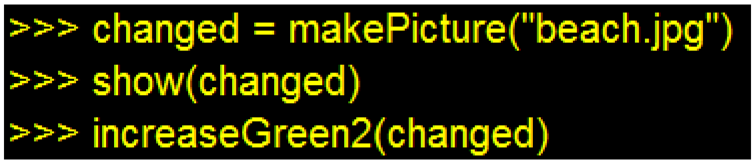
Remember how we looked at this last week by writing something like this?

```
def increaseGreen2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      pixel = getPixel(picture,x,y)
      greenValue = getGreen(pixel)
      setGreen(pixel, greenValue*1.5)
      repaint(picture)
```

Remember how this took FOREVER? We actually can speed it up a little bit by writing this instead (notice the indentation of the statement **repaint(picture)**:

```
def increaseGreen2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      pixel = getPixel(picture,x,y)
      greenValue = getGreen(pixel)
      setGreen(pixel, greenValue*1.5)
    repaint(picture)
```

This modifies the code so that instead of repainting **after every pixel is changed**, it repaints **after each column is changed**. Try it to see for yourself.



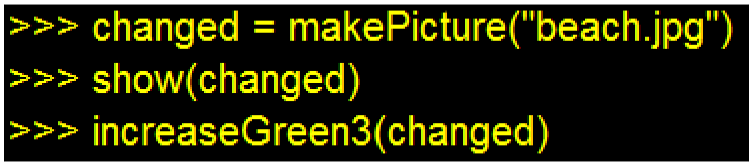
```
>>> changed = makePicture("beach.jpg")
>>> show(changed)
>>> increaseGreen2(changed)
```

[Q7] What does it look like when we run this?

By the way, the order we write the two loops has an effect on the order that the pixels are painted. Add this to your file:

```
def increaseGreen3(picture):
  for y in range(0,getHeight(picture)):
    for x in range(0,getWidth(picture)):
      pixel = getPixel(picture,x,y)
      greenValue = getGreen(pixel)
      setGreen(pixel, greenValue*1.5)
      repaint(picture)
```

I just said that when the x loop is the outer loop that we have a column wise function. This code has y as the outer loop. Try it.



```
>>> changed = makePicture("beach.jpg")
>>> show(changed)
>>> increaseGreen3(changed)
```

[Q8] What does it look like when we run this?