More with Pictures and Loops

In the previous section, we saw that we can use a `for` loop to access every pixel in a picture and then do something to it. In many applications, we don't need to access each of the pixels, but rather would like to do something to just a few select pixels. Removing the redeye from a picture would be a perfect example of this. In this case, we would only need to change the color of the pixels in a small region of the picture. To do this in Python, we are going to use the `range` function to get a list of numbers that we will use to get the coordinates of the pixels we want to modify.

There are three ways to use the `range` function in Python – with one, two, or three parameters. Let's look at examples to show how these different versions of `range` work.

Example 1: The `range` function

1-parameter range:
$print(range(3))$
[0, 1, 2]
$print(range(10))$
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

2-parameter range:
$print(range(1,4))$
[1, 2, 3]
$print(range(8, 15))$
[8, 9, 10, 11, 12, 13, 14]

3-parameter range:
$print(range(1, 6, 2))$
[1, 3, 5]
$print(range(0, 10, 2))$
[0, 2, 4, 6, 8]
$print(range(1, 20, 3))$
[1, 4, 7, 10, 13, 16, 19]
$print(range(10, 0, -1))$
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

The 1-parameter `range` function returns a list of integers that starts at 0 and goes up to the value that was input, but does not include it. The 2-parameter `range` function returns a list of integers that starts at the first number indicated and goes up to, but does not include, the second number. The 3-parameter `range` function works similarly to the 2-parameter `range` function, except that the third parameter specifies what the step is (*i.e.*, what to count by). Notice that in the last example of

the 3-parameter `range` function, the range starts at 10 and goes *down* to 0. The step is negative to make the count go backwards.

The following function to calculate the sum of the first *n* integers uses the range function to give a list of numbers to be used in a loop:

Example 2: Function to sum first n integers

```
def sumIntegers(n):
  sum = 0
  for j in range(1, n + 1):
    sum = sum + j
  return sum
```

How does this work? The variable *sum* will hold the accumulated total as we go through the loop; it starts out with the value 0. The first time we go through the loop, *j* takes on the value 1. It then gets added to the value in *sum* (currently 0), and then the result gets stored back in *sum*. So *sum* now has the value 1. The next time through the loop, *j* takes on the next value in the list, which is 2. This value gets added to what is in *sum* (currently 1), and gets stored back in *sum*. So *sum* now has the value 3. This continues until the last iteration of the loop, when *x* takes on the value of *n*. This gets added to the value in *sum*, and stored back in *sum*. The variable *sum* then contains the value of 1+2+3+…+*n*.

Now how can we use this `range` function to do anything with pictures? The basic idea is that we will use it in our loop statements to indicate which coordinates of pixels we would like to modify. Consider the following example that will change the color of the first 20 pixels in the 5th row of the picture to black.

Example 3: Change some pixels to black

```
def makeSomeBlack(picture):
  # Make a copy of picture to work with
  newPict = duplicatePicture(picture)

  # change the pixels
  for x in range(20):
    px = getPixel(newPict, x, 5)
    setColor(px, black)

  # return the new picture
  return newPict
```

So how does this work? We begin by copying our original picture so that we work with a copy, not the original. Then we loop through the first 20 integers (0 through

19). Each time through the loop, we get one pixel, with coordinates (x, 5) and change the color of that pixel to black. The first time through the loop, we get and change pixel (0,5). The second time through the loop, we get and change pixel (1, 5). The next time through the loop, we get and change pixel (2, 5). We continue this until the last time through the loop, when we get and change pixel (19, 5).

Suppose we wanted to change the entire row to black. We only need to modify the parameter in the `range` function of the example so that it ends at the last pixel in the row (instead of the 20th pixel in the row). If our picture is 640 x 480, the last pixel in row 5 is (639, 5). We will use the `getWidth` function to specify what this should be.

Example 4: Change pixels in row 5 to black

```
def changeRow5(picture):
  # Make a copy of picture to work with
  newPict = duplicatePicture(picture)

  # change the pixels
  for x in range(getWidth(newPict)):
    px = getPixel(newPict, x, 5)
    setColor(px, black)

  # return the new picture
  return newPict
```

Notice that the only difference between this example and Example 3 is the parameter in the `range` function.

Let's extend this example so that we now change the color of *every* pixel in the picture to black. We could write functions similar to `changeRow5` such as `changeRow0`, `changeRow1`, `changeRow2`, etc. This is really too much work! Instead of doing this, we will actually use a loop *inside* of a loop (nested loops). If we want to change all of the pixels in all of the rows and all of the columns, we will use one loop to color each row like we did with `changeRow5`, and then we will next that loop inside of a loop that lets us go through each of the rows. The code would look like the following:

Example 5: Color all pixels using nested loops

```
def colorAllBlack(picture):
  # Make a copy of picture to work with
  newPict = duplicatePicture(picture)

  # change the pixels
  for y in range(getHeight(newPict)):
    for x in range(getWidth(newPict)):
      px = getPixel(newPict, x, 5)
      setColor(px, black)

  # return the new picture
  return newPict
```

How does this work? For *each* value of *y* in the list [0, …, h-1] (where *h* is the height of the picture), we go through the entire loop for *x*. So *y* starts at 0. Then *x* starts at 0 and we modify pixel (0, 0). Then *y* stays at 0 and *x* becomes 1. We then modify pixel (1, 0). We modify all of the pixels (2, 0), (3, 0), …, (w-1, 0) before *y* changes value. After we modify pixel (w-1, 0), *y* becomes 1, and we modify pixel (0, 1). We then modify all the pixels (1, 1), (2, 1), (3, 1), …(w-1, 1). Then *y* will become 2 and we will modify pixels (0, 2), (1, 2), (2, 2), (3, 2), …etc. We continue this until *y* becomes h-1 and we modify the pixels in the bottom row of the picture: (0, h-1), (1, h-1), (2, h-1), …, (w-1, h-1). The value of *y* tells us which row we are working with, and the values of *x* let us move right across the picture.

We should now practice these ideas of using the range function to get coordinates of pixels to modify by working through the next mini-lab.

Mini-Lab: More practice with for loops for manipulating pixels in a picture