Conditional Statements

In previous examples and exercises, we have been manipulating images by using nested for-loops to iterate over all of the pixels in an image, or we have modified the ranges of the loops to focus on just one portion of an image. Sometimes we would like to apply a change to all of the pixels in an image that have some particular property.

Fortunately for us, there is a way to "pick out" the pixels that we want to change. We will use an `if`-statement in Python to allow us to make selections. `If`-statements are *conditional statements* – if some given condition is true, we execute the block of code that follows. The general syntax is:

```
if some condition:
```

where some condition is replaced by an expression that has a true or false value. This could be an expression such as `x < 100`, or `redValue > 255`, or `x + y > 200`. The block of code that would follow this statement is whatever you want to have happen when the condition is true.

We can use the logical operators <, <=, >, >=, ==, !=, and <> when constructing expressions. We use the double equals to test for equality. (Remember, the single equal symbol is used for assignment of values to variables.) We can use either != or <> to test non-equivalence.

We can make more complicated expressions by using the keywords `and` and `or` to combine expressions, such as in the expression:

```
x > 100 and x < 200.
```

Example 1: Using if-statements to draw a diagonal line.
To draw a line diagonally from the upper left corner to the bottom right corner of a square image, we would need to change the color of the pixels along the diagonal. These are the pixels (0,0), (1, 1), (2, 2), ...(*w-1, w-1*). Notice, these pixels have the same *x*- and *y*- values. When we use the nested loops to iterate over all of the pixels in the image, we can check if the x-value is the same as the y-value of each pixel. When they are equal, we change the color of the pixel. The code to do this is shown below:

```
# Draw a diagonal from upper-left to lower-right
def drawDiagonal(picture):
  newPic = picture.copy()

  for y in range(newPic.height):
    for x in range(newPic.width):
```

```
      if x == y:
        newPic.putpixel((x, y), (255, 0, 0))
```

If you test this example with pictures of various sizes, you will find that the pictures also do not need to be square. In the case of non-square images, the diagonal is not a true diagonal from the top left corner to the bottom right corner.

We can look at several other examples of using if-statements to make more sophisticated color manipulations.

Example: TintRed

```
def tintRed(picture):
  newPic = picture.copy()
  for y in range(newPic.height):
    for x in range(newPic.width):
      rvalue, gvalue, bvalue = newPic.getpixel((x,y))
      if (rvalue < 100):
        newR = int(rvalue * 2.5)
        newPic.putpixel((x,y), (newR, gvalue, bvalue))
  return newPic
```

Exercise: What does this function do to a picture?

Next, creating sepia-tones is a way to give a picture an old-fashioned look, with a yellowish tint. The picture first gets converted to grayscale (because older prints were usually in grayscale, and they are easier to work with), and then we look for high and low ranges of color (luminance) and change them separately. The values used in this example can be tweaked if you'd like to change the effect.

Original image

Sepia-tinted image

The function to create this effect is on the following page.

Example: Creating sepia-tinted images

```
def sepiaTint(picture):
  newPic = grayscale(picture)
  for y in range(newPic.height):
    for x in range(newPic.width):
      rvalue, gvalue, bvalue = newPic.getpixel((x,y))

      # tint shadows
      if (rvalue < 63):
        rvalue = int(rvalue * 1.1)
        bvalue = int(bvalue * 0.9)

      # tint midtones
      if (rvalue > 62 and rvalue < 192):
        rvalue = int(rvalue * 1.15)
        bvalue = int(bvalue * 0.85)

      # tint highlights
      if (rvalue > 191):
        rvalue = int(rvalue * 1.08)
        bvalue = int(bvalue * 0.93)

      # set the new color of the pixel
      newPic.putpixel((x,y), (rvalue, gvalue, bvalue))

  # return the new image
  return newPic
```

Notice in this sepia-tint example that there are three sets of `if`-statements, one to modify the red and blue values for each of the different ranges of red.

There are times when we may want to do an alternative action if our condition is not true. In those cases, we will use an `if-else` statement.

As an example, we might make a simple posterized effect. Printed posters often have a limited number of colors that can be used, so we may want to posterize a picture before having it printed. To do this, we look for specific ranges of colors, and then set the color values in each range to *one* particular value.

Example: Simple Posterize

```
def simplePosterize(picture):
  newPic = picture.copy()

  for y in range(newPic.height):
    for x in range(newPic.width):
      rvalue, gvalue, bvalue = newPic.getpixel((x,y))

      # check red
      if (rvalue > 128):
        newR = 240
      else:
        newR = 50

      # check green
      if (gvalue > 128):
        newG = 240
      else:
        newG = 50

      # check blue
      if (bvalue > 128):
        newB = 240
      else:
        newB = 50

      # Set new color of pixel
      newPic.putpixel((x,y), (newR, newG, newB))

  # return the new image
  return newPic
```

Original picture

Simple posterized picture

We can extend this posterize example by choosing to have more than two choices for red, green, and blue. To do this, we will use if-else if-else statements, which are abbreviated as `elif` statements in Python. We have a more general posterize example:

Example: Posterize

```
def posterize(picture):
  newPic = picture.copy()

  for y in range(newPic.height):
    for x in range(newPic.width):
      rvalue, gvalue, bvalue = newPic.getpixel((x,y))

      # check red
      if (rvalue < 64):
        newR = 31
      elif (rvalue > 63 and rvalue < 128):
        newR = 95
      elif (rvalue > 127 and rvalue < 192):
        newR = 191
      else:
        newR = 223

      # check green
      if (gvalue < 64):
        newG = 31
      elif (gvalue > 63 and gvalue < 128):
        newG = 95
      elif (gvalue > 127 and gvalue < 192):
        newG = 191
      else:
        newG = 223

      # check blue
      if (bvalue < 64):
        newB = 31
      elif (bvalue > 63 and bvalue < 128):
        newB = 95
      elif (bvalue > 127 and bvalue < 192):
        newB = 191
      else:
        newB = 223
```

```
      newPic.putpixel((x,y), (newR, newG, newB))
  return newPic
```

How does this function work? Once we obtain the red, green, and blue values for a pixel, we check the red values. We come to the first `if`-statement. If the given condition (`rvalue` < 64) is true, we set the red to a new value (31) and then execution continues to the `if`-statement checking the green values. We skip the remaining statements in that block of code dealing with red values. If the first condition for red is not true, we would check the second condition (`rvalue` > 63 and `rvalue` < 128). If it is true, we change the red value to 95 and execution skips to the first `if`-statement for green values. If it is not true, we check the third condition. If it is true, we change the red value to 159. If it is not true, we execute the statement(s) that come after the `else`.

Exercise: What is the minimum number of conditions we would need to check in the `posterize` example? What is the maximum number of conditions we would need to check?

Exercise: What is the difference between using all `if`-statements in the following version of posterize and the version from earlier in the notes?

```
def posterize2(picture):
  newPic = picture.copy()

  for y in range(newPic.height):
    for x in range(newPic.width):
      rvalue, gvalue, bvalue = newPic.getpixel((x,y))

      # check red
      if (rvalue < 64):
        newR = 31
      if (rvalue > 63 and rvalue < 128):
        newR = 95
      if (rvalue > 127 and rvalue < 192):
        newR = 191
      if (rvalue > 191 and rvalue < 256):
        newR = 223

      # check green
      if (gvalue < 64):
        newG = 31
      if (gvalue > 63 and gvalue < 128):
        newG = 95
      if (gvalue > 127 and gvalue < 192):
```

```
        newG = 191
      if (gvalue > 191 and gvalue < 256):
        newG = 223

      # check blue
      if (bvalue < 64):
        newB = 31
      if (bvalue > 63 and bvalue < 128):
        newB = 95
      if (bvalue > 127 and bvalue < 192):
        newB = 191
      if (bvalue > 191 and bvalue < 256):
        newB = 223

      # Set new color of pixel
      newPic.putpixel((x,y), (newR, newG, newB))

  # return the new image
  return newPic
```

Review:

`If`-statements have the following syntax:

```
if some condition:
  # do something here if the condition is true
```

`If-else` statements have the following syntax:

```
if some condition:
  # do something here if the condition is true
else:
  # do something else if the condition is not true
```

`Elif` statements have the following syntax:

```
if some condition:
  # do something here if the condition is true
elif some other condition:
  # do something else if the first condition is not true
  # and the second condition is true
elif yet another condition:
  # do if the first two conditions are not true but the
```

```
  # third condition is true
else:
  # do when none of the conditions are true
```

Note that there may be *any* number of `elif` conditions.

We will now experiment with using if-statements in the next activity.

Activity: Experimenting with Conditional Statements