Conditional Statements

In an earlier mini-lab, when you were drawing a diagonal line on a picture, you probably noticed that the coordinates of the pixels on the diagonal had the same x- and y- values. At the time we were drawing these lines, we were using nested for-loops to give us the x- and y-coordinates of the pixels. When we got to this example, we found that we only needed one loop – once we had the value of x, we also had the value of y. We could have used the nested loops if we had a way to "pick out" the correct values of x and y – that is, pick them out when they were equal.

Fortunately for us, there is a way to "pick out" values that we want. 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.
```

Returning to our opening example, let's use if-statements to help us draw a diagonal line:

```
# Draw a diagonal from upper-left to lower-right
def drawDiagonal(picture):
  newPict = duplicatePicture(picture)
  for y in range(getHeight(newPict)):
    for x in range(getWidth(newPict)):
      if x == y:
        px = getPixel(newPict, x, y)
        setColor(px, red)
  return newPict
```

If you test this example with pictures of various sizes, you will find that the pictures also do not need to be square.

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

Example: TintRed

```
def tintRed(picture):
  newPict = duplicatePicture(picture)
  for px in getAllPixels(newPict):
    redValue = getRed(px)
    if redValue < 100:
      newRedValue = redValue * 2.5
      setRed(px, newRedValue)
  return newPict
```

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.

```
def sepiaTint(picture):
  newPict = grayscale(duplicatePicture(picture))

  for px in getAllPixels(newPict):
    redValue = getRed(px)
    blueValue = getBlue(px)

    # tint shadows
    if (redValue < 63):
      redValue = redValue * 1.1
      blueValue = blueValue * 0.9

    # tint midtones
    if (redValue > 62 and redValue < 192):
      redValue = redValue * 1.15
      blueValue = blueValue * 0.85

    # tint highlights
    if (redValue > 191):
      redValue = redValue * 1.08
      blueValue = blueValue * 0.93

    # set the new colors
```

```
    setRed(px, redValue)
    setBlue(px, blueValue)

  return newPict
```


Original picture


Sepia-tinted picture

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 and blue.

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):
  newPict = duplicatePicture(picture)

  for px in getAllPixels(newPict):
    redValue = getRed(px)
    greenValue = getGreen(px)
    blueValue = getBlue(px)

    # check and set red values
    if (redValue > 128):
      setRed(px, 240)
    else:
      setRed(px,50)

    # check and set green values
    if (greenValue > 128):
```

```
      setGreen(px, 240)
    else:
      setGreen(px, 50)

    # check and set blue values
    if (blueValue > 128):
      setBlue(px, 240)
    else:
      setBlue(px, 50)

  return newPict
```

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):
  newPict = duplicatePicture(picture)

  for px in getAllPixels(newPict):
    # get the RGB values
    redValue = getRed(px)
    blueValue = getBlue(px)
    greenValue = getGreen(px)

    # check and set red values
    if (redValue < 64):
      setRed(px, 31)
    elif (redValue > 63 and redValue < 128):
      setRed(px, 95)
```

```
    elif (redValue > 127 and redValue <192):
      setRed(px, 159)
    else:
      setRed(px, 223)

    # check and set green values
    if (greenValue < 64):
      setGreen(px, 31)
    elif (greenValue > 63 and greenValue < 128):
      setGreen(px, 95)
    elif (greenValue > 127 and greenValue <192):
      setGreen(px, 159)
    else:
      setGreen(px, 223)

    # check and set blue values
    if (blueValue < 64):
      setBlue(px, 31)
    elif (blueValue > 63 and blueValue < 128):
      setBlue(px, 95)
    elif (blueValue > 127 and blueValue <192):
      setBlue(px, 159)
    else:
      setBlue(px, 223)

  return newPict
```

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 (`redValue` < 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 (`redValue` > 63 and `redValue` < 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):
  newPict = duplicatePicture(picture)

  for px in getAllPixels(newPict):
    # get the RGB values
    redValue = getRed(px)
    blueValue = getBlue(px)
    greenValue = getGreen(px)

    # check and set red values
    if (redValue < 64):
      setRed(px, 31)
    if (redValue > 63 and redValue < 128):
      setRed(px, 95)
    if (redValue > 127 and redValue <192):
      setRed(px, 159)
    if (redValue > 191 and redValue < 256):
      setRed(px, 223)

    # check and set green values
    if (greenValue < 64):
      setGreen(px, 31)
    if (greenValue > 63 and greenValue < 128):
      setGreen(px, 95)
    if (greenValue > 127 and greenValue <192):
      setGreen(px, 159)
    if (greenValue > 191 and greenValue < 256):
      setGreen(px, 223)

    # check and set blue values
    if (blueValue < 64):
      setBlue(px, 31)
    if (blueValue > 63 and blueValue < 128):
      setBlue(px, 95)
    if (blueValue > 127 and blueValue <192):
      setBlue(px, 159)
    if (blueValue > 191 and blueValue < 256):
      setBlue(px, 223)

  return newPict
```

We have started to see that we can use `if`-statements to help us control how we make our color changes. We can continue with this idea by replacing one color with another color. Whenever the color of a pixel is "close enough" to the original color, we replace it. In a sense, we are looking at the "distance" between colors of pixels. We will use this to remove red-eye in a picture as well as to replace backgrounds.

Example: Remove red-eye

```
def removeRedEye(pic, startX, startY, endX, endY,
replacementColor):
  newPict = duplicatePicture(pic)
  for x in range(startX, endX):
    for y in range(startY, endY):
      px = getPixel(newPict, x, y)
      if (distance(red, getColor(px)) < 165):
        setColor(px, replacementColor)
  return newPict
```

In the following picture, I used the openPictureTool in jes4py to find the x- and y-ranges for the eyes.

In `main`, the `removeRedEye` function was called as follows:

```
pict2 = removeRedEye(pict, 170, 180, 193, 196, makeColor(30,30,50))
pict3 = removeRedEye(pict2, 238, 184, 256, 198, makeColor(30,30,50))
```

Original Picture

Red-eye removed

After experimenting a little, a replacement color that is a deep shade of blue was found to look fairly reasonable.

Aside: Distance Function

The distance function used to compare colors is similar to that seen in mathematics. Recall that you can find the distance between two points in space by using the following formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

To compute the distance between the colors of two pixels, the following formula is used:

$$d = \sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2},$$

where (r_1, g_1, b_1) and (r_2, g_2, b_2) are the RGB values of the two pixels.

We can also use the distance function to aid in replacing backgrounds. Suppose we have a picture of someone and a picture of where they stood without them. Could we subtract the background of the person and then replace another background? We would have to figure out where the colors of the two pictures are exactly the same (i.e., the backgrounds should have the same colors in the regions where the person is not standing.)

We can use the following function to attempt to do this:

Example: Swap backgrounds by subtraction

```
# replace the background around a person with a new
# background
def swapBack(bgWithPerson, bgWithoutPerson, newBg):
  newPict = duplicatePicture(bgWithPerson)
  for y in range(getHeight(newPict)):
    for x in range(getWidth(newPict)):
      px1 = getPixel(newPict, x, y)
      bgpx = getPixel(bgWithoutPerson, x, y)
      if (distance(getColor(px1), getColor(bgpx)) < 15.0):
        setColor(px1, getColor(getPixel(newBg, x, y)))
  return newPict
```

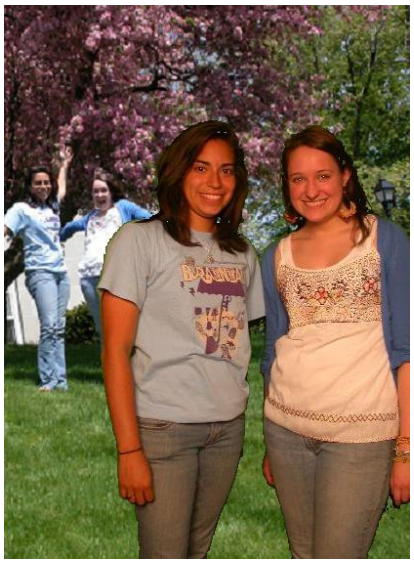
In this function, the parameter `bgWithPerson` is a picture with a person in front of some background, the parameter `bgWithoutPerson` is a picture of just that background, and the parameter `newBg` is some new background, like the Eiffel Tower. When we compare the colors of corresponding pixels in the `bgWithPerson` and the `bgWithoutPerson` pictures, the difference between these colors should be very small, except where the person is. So we would check where that distance is small, and then replace the color of those pixels with the color from the pixels in the new background.

Television producers will sometimes use an effect called *chromakey*. This is very common with weather forecasters. The idea is that the person will stand in front of

a solid color background (typically green or blue) and then the background will be replaced with some other image, like a map or important building. In the following example, we have two girls in front of a green screen, and then a picture of them on campus. The green background then gets replaced with the campus picture.

green-screen picture

background picture



Result of chromakey

The following function was used to produce this new image. Depending on the actual shade of green in the green screen picture, the definition of green may need to be altered.

```
# replace the green background with a more interesting
# background
def chromakey(pict, bg):
  newPict = duplicatePicture(pict)
  for y in range(getHeight(newPict)):
```

```
    for x in range(getWidth(newPict)):
      px = getPixel(newPict, x, y)
      # A definition of green
      if (getRed(px) + getBlue(px) < getGreen(px)):
        # then grab the color at the same spot from the bg
        setColor(px, getColor(getPixel(bg, x, y)))
  return newPict
```

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 minilab and lab.

Mini-Lab: Selectively Changing Colors

Lab: Combining Pictures