CS Kickstart – Day 2

Defining Functions

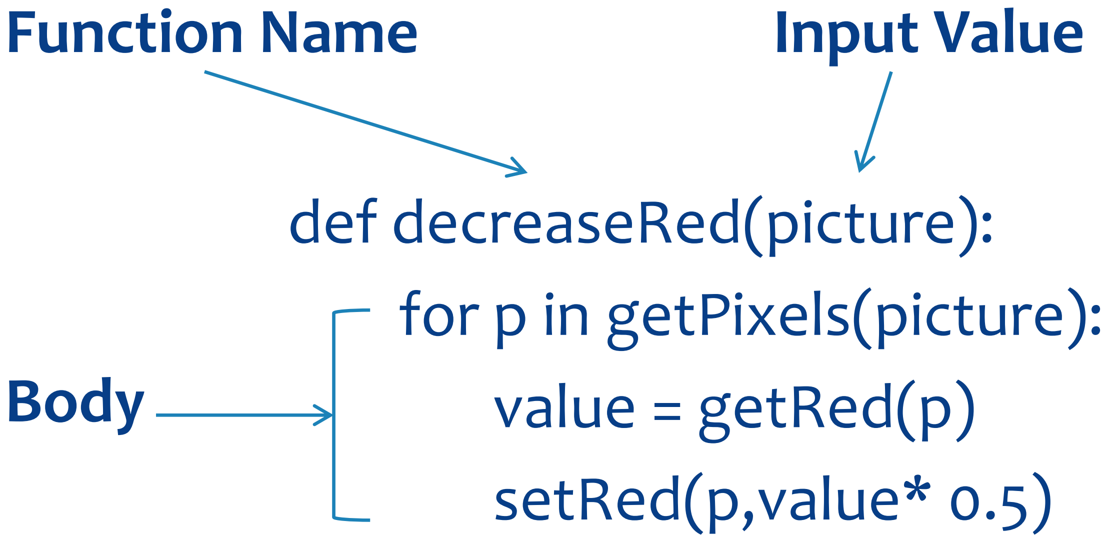
Conditionals (If Statements)

Double For Loops

Review: Parts of a Function

- Function Name
- Input Values (optional)
  - Zero, one, or many
  - Sometimes called “parameters”
- Function Body
  - Indented

Review: Parts of a Function



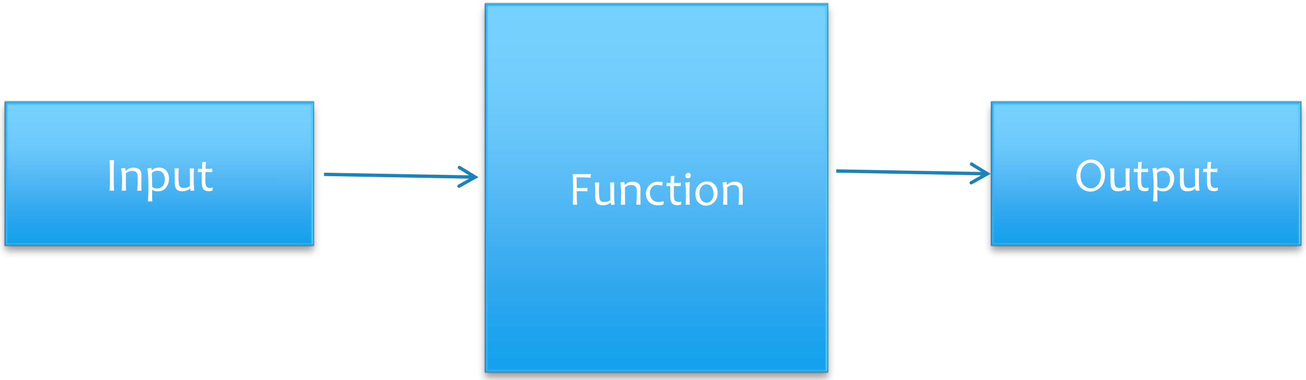
Function Name

Input Value

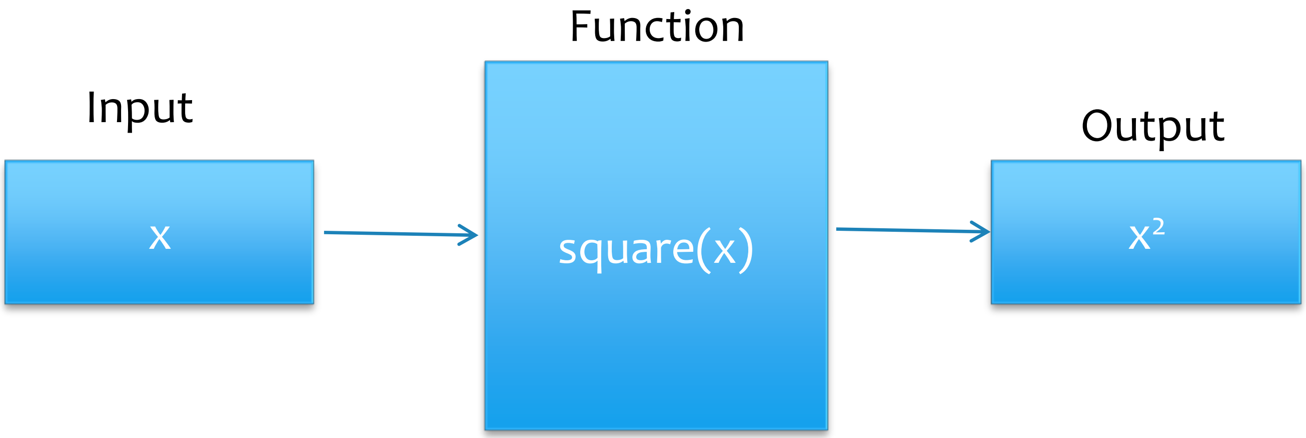
Body

```
def decreaseRed(picture):
    for p in getPixels(picture):
        value = getRed(p)
        setRed(p,value* 0.5)
```

What do functions do?



What do functions do?



Function

Input

Output

x

square(x)

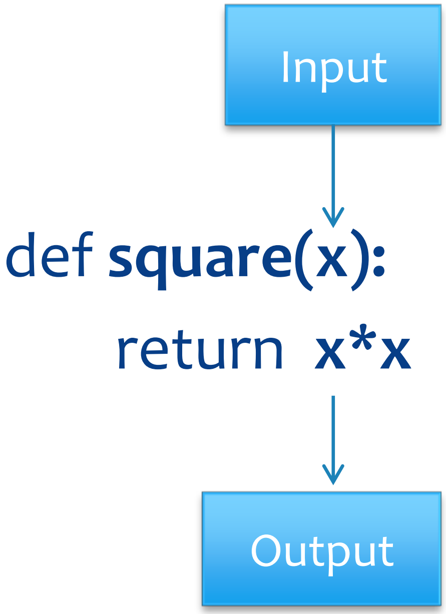
x^2

Calling a Function

```
def square(x):
      return  x*x

y = square(4)
print y
```

What would print?



```
def square(x):
      return  x*x
```

Calling a Function

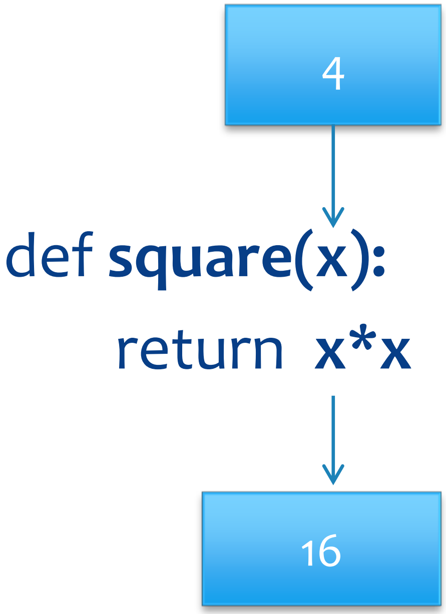
```
def square(x):
    return  x*x
```

y = **square**(4)

y = output of function “square”

called with input value 4

y = 16



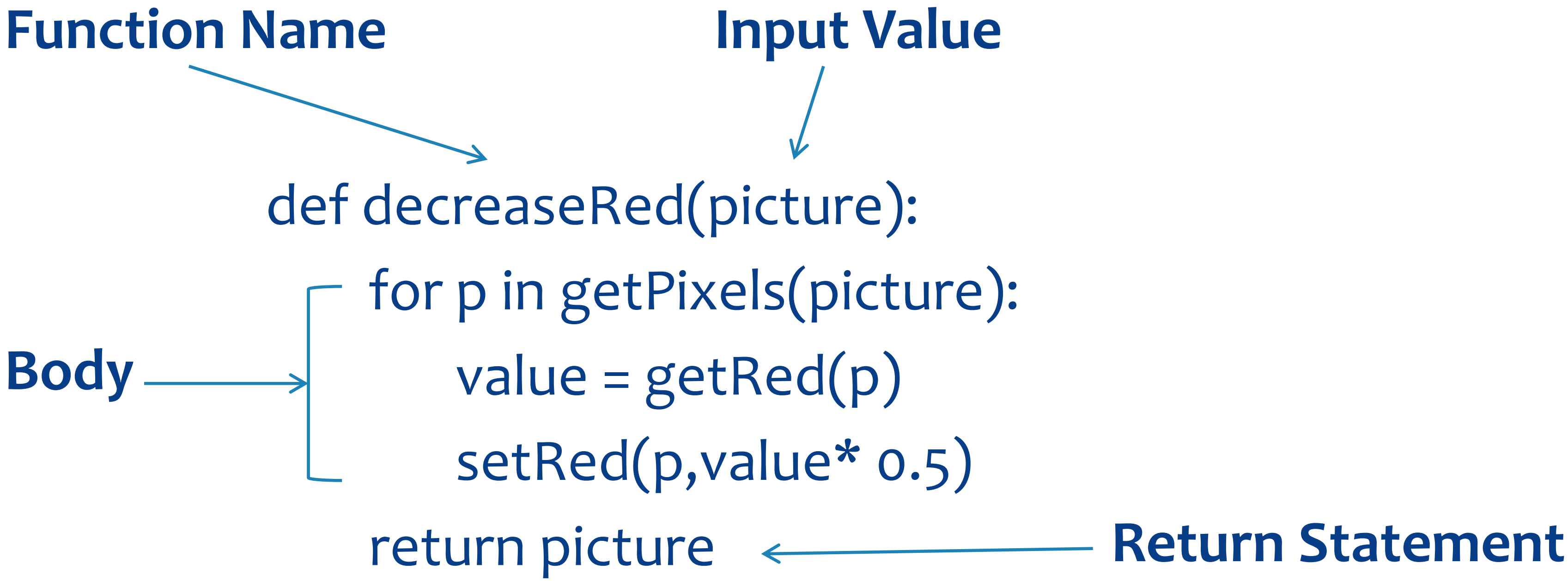
4

def **square(x):**

return **x*x**

16

Parts of a Function



Function Name

Input Value

```
def decreaseRed(picture):
    for p in getPixels(picture):
        value = getRed(p)
        setRed(p,value* 0.5)
    return picture
```

Body

Return Statement

```
def decreaseRed(picture):
    for p in getPixels(picture):
        value = getRed(p)
        setRed(p,value* 0.5)
    return picture
```

return means output

result = output

```
>>> result = decreaseRed(myPicture)
>>> show(result)
```

How ‘if’ Statements Work

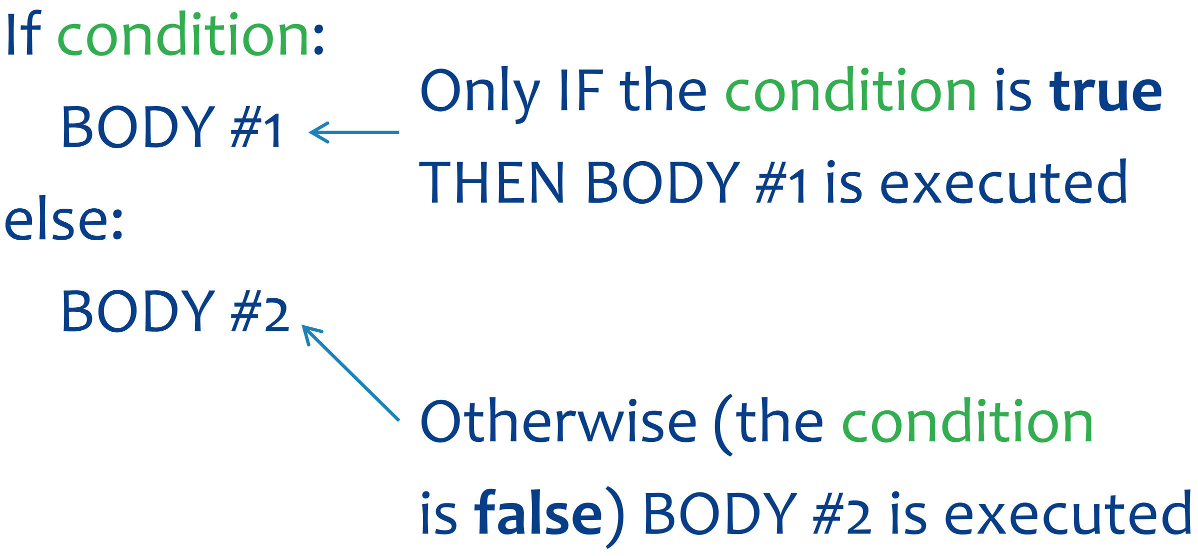
```
If condition:
    BODY #1
```

← Only IF the condition is **true** THEN BODY #1 is executed

```
x = 2
If x < 2 :
    print “Body 1”
```

- Is something printed?

How ‘if’ Statements Work



```
If condition:
    BODY #1
else:
    BODY #2
```

Only IF the condition is **true** THEN BODY #1 is executed

Otherwise (the condition is **false**) BODY #2 is executed

```
x = 2
If x < 2 :
    print “Body 1”
else:
    print “Body 2”
```

- What is printed?

Conditionals

Conditions need to evaluate to **true** or **false.**

Here are other conditions you can use:

- Test equality with ‘==‘
  - Ex: 4 == 5
- Text not equal with ‘!=‘
  - Ex: 4 != 5
- Test comparisons > , >= , < , <=
- and , or

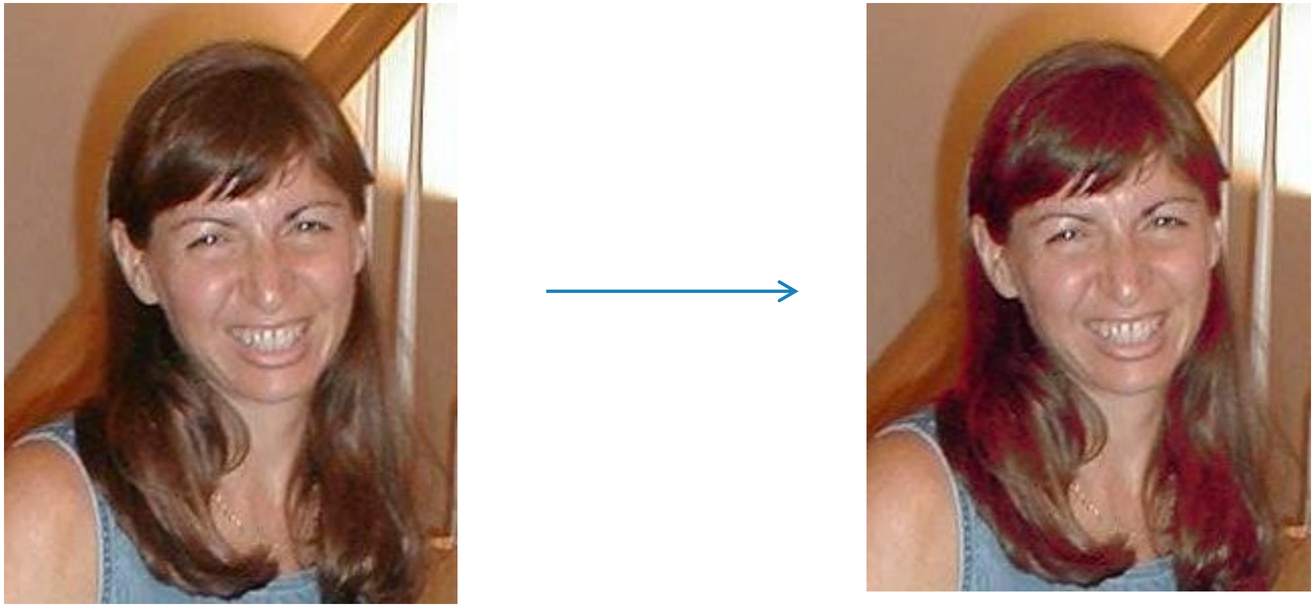
‘if’ in action

```
if  x < 10:
    print “small”

if  x >= 10 and x < 20:
    print “medium”

if x >= 20 and x < 30:
    print “large”
```

Let’s Make Barbara a Redhead

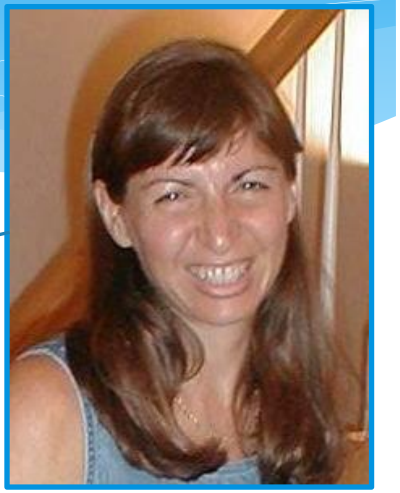


Why can’t we use our increaseRed function?

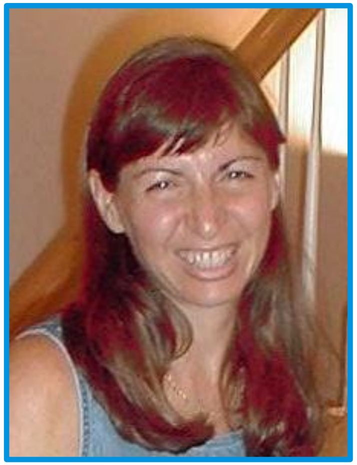
Psuedocode for Red Hair

```
def turnRed():
    make a picture using a file
    for each pixel in the picture:
        figure out the color of the pixel
        if the color is close to the brown color in her hair:
            increase the redness of this pixel
    show the picture
    return the picture
```

Implementation



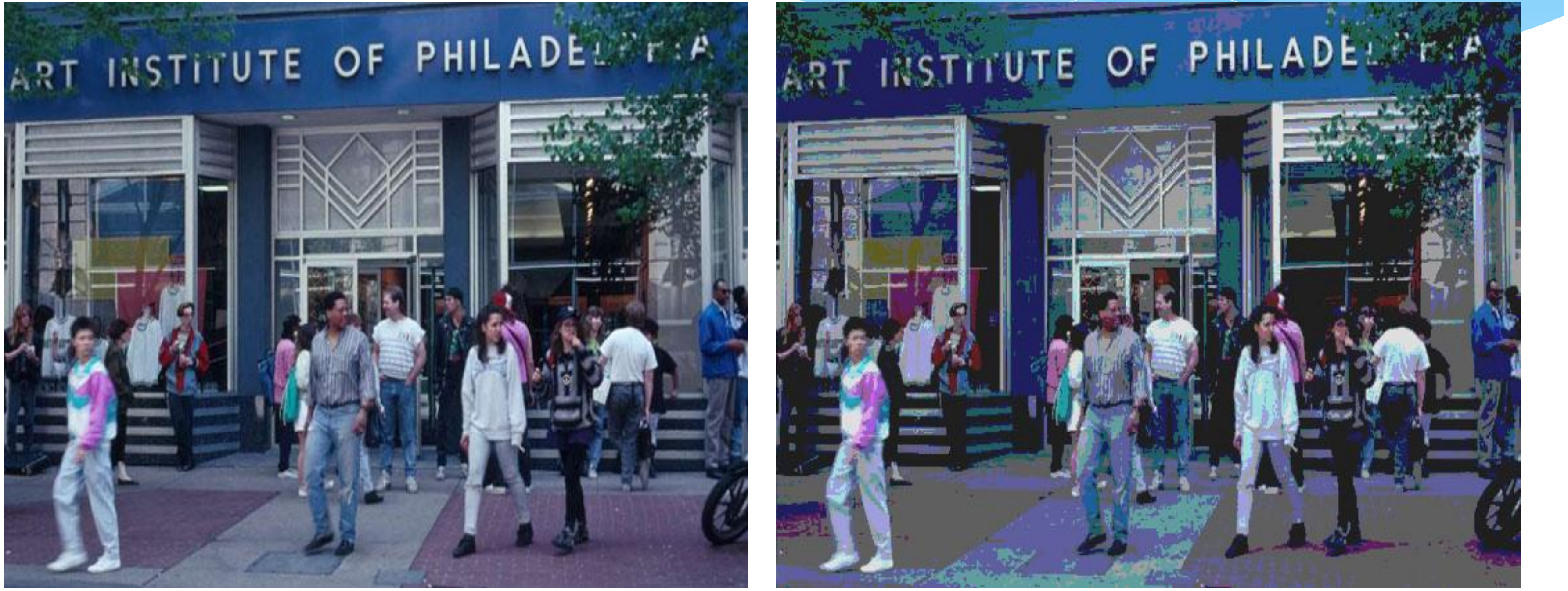
```
def turnRed():
    make a picture using a file
    for each pixel in the picture:
        figure out the color of the pixel
        if the color is close to the brown color in her hair:
            increase the redness of this pixel
    show the picture
    return the picture
```



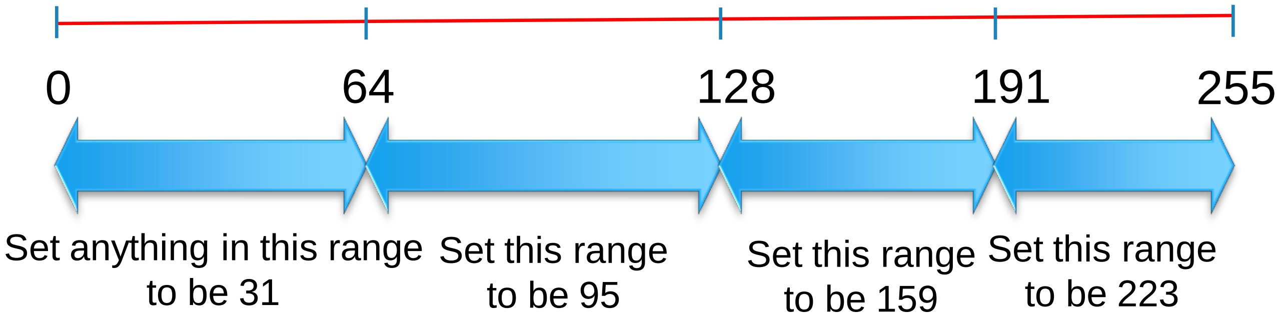
Let’s do exercise #1 !

If you finish exercise #1, feel free to move on to exercise #2. In a minute, we will go over hints for how to complete exercise #2.

Posterizing:
Reducing the range of colors



Posterizing: How we do it



0 64 128 191 255

Set anything in this range to be 31

Set this range to be 95

Set this range to be 159

Set this range to be 223

- *Range* of colors maps to a *single* color
- **If** statements to find which range

- End result: *many* colors → *few* colors

Posterizing Psuedocode

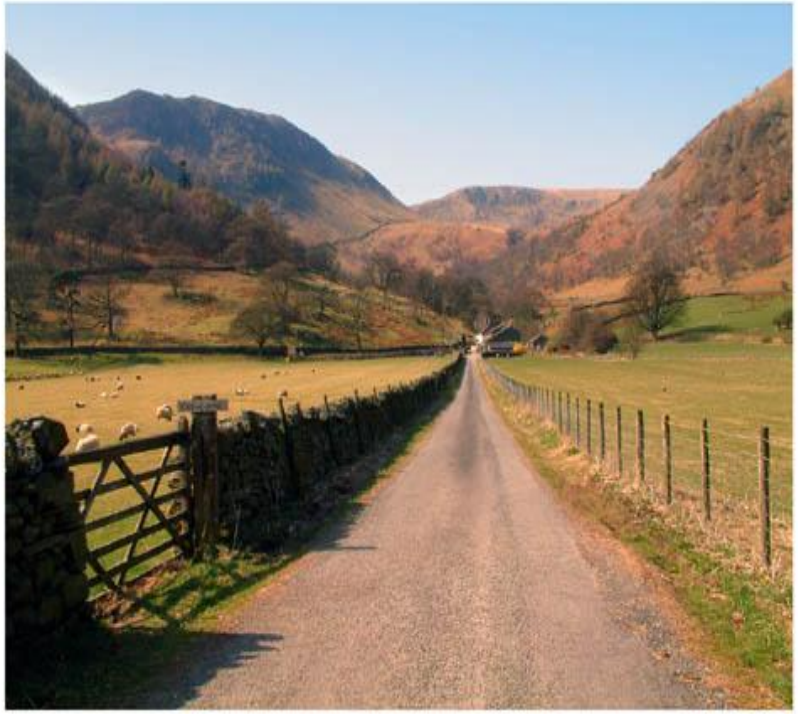
```
def posterize(picture):
        for each pixel in the picture:
            get the red value of the pixel and call it redValue
            if redValue is less than 64:
                set red of pixel to be 31
            if redValue is between 64 and 128:
                set red of pixel to be 95
```

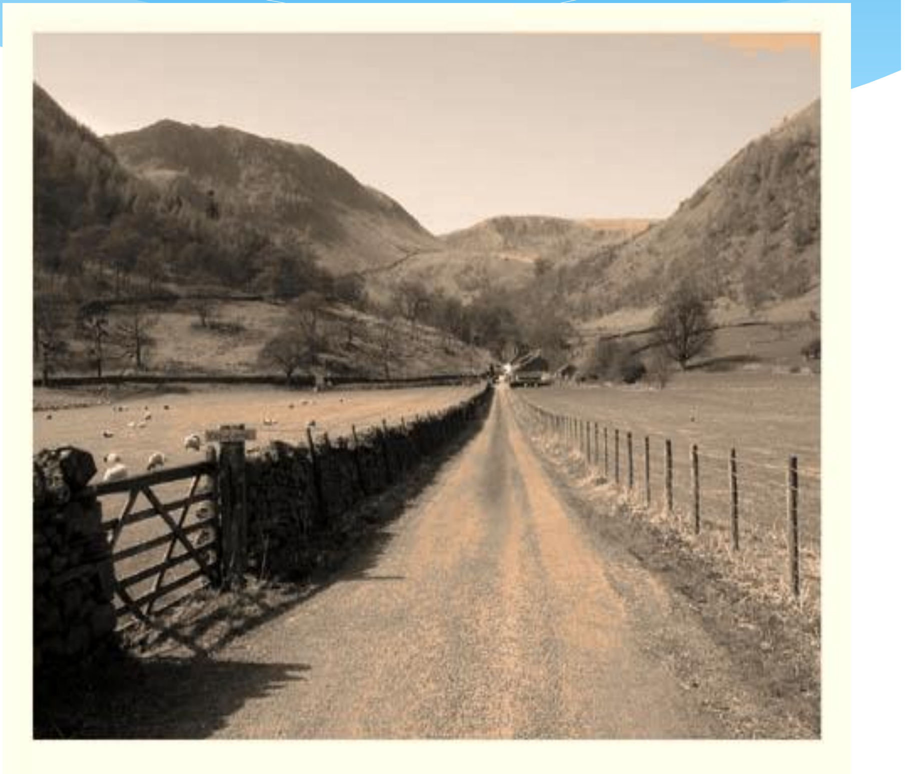
```
if (redValue > 63 and redValue < 128)
```

Exercise #2

If you finish exercise #2, you may move on to #3. We will break in a minute to talk about hints for exercise #3.

Exercise #3: Sepia-toned prints





Generating sepia-toned prints

- Yellowish tint that we associate with older photographs.
- Can’t just increase the amount of yellow
- Range of colors converted to other colors
  - We can create such conversions using if

Here’s how we do it

```
def sepiaTint(picture):
    Convert image to greyscale
    Loop through pixels to tint each pixel
        find red and blue values of pixel
        tint shadows
        tint midtones
        tint highlights
        set new pixel color values for red and blue
```

Double For loops and Ranges

This will help you complete exercise #4

Accessing Each Pixel

How do we access pixels?

```
for pixel in getPixels(picture): ← seems like magic
        do something to pixel
```

Is there another way?

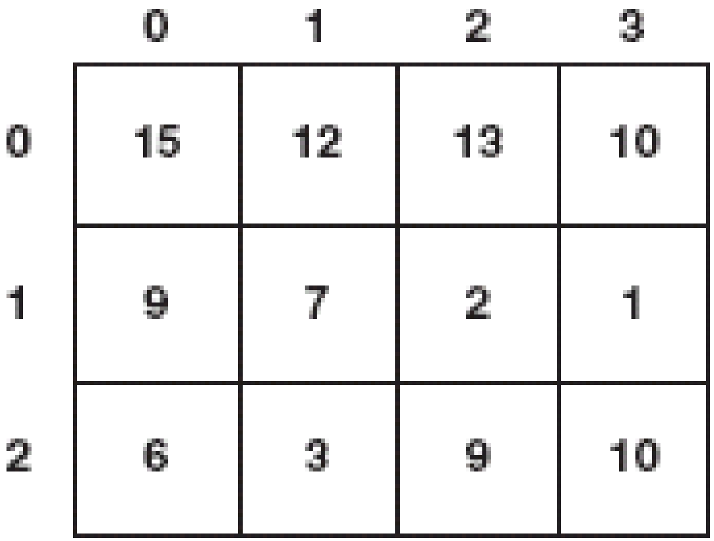
A Picture is a *matrix* of pixels

- A continuous line is an *array*
  - *1* dimension
- Pictures have 2 dimensions
  - Height
  - Width
- Our array needs 2 dimensions
  - a *matrix*

0	1	2	3
15	12	13	10

	0	1	2	3
0	15	12	13	10
1	9	7	2	1
2	6	3	9	10

Referencing a matrix



	0	1	2	3
0	15	12	13	10
1	9	7	2	1
2	6	3	9	10

- We talk about positions in a matrix as (x,y), or (horizontal, vertical)
- Element (1,0) in the matrix at left is the value 12
- What is the value of element (0,2) ?

How to access each pixel

x loops up to width

Y loops up to height

	0	1	2	3
0	15	12	13	10
1	9	7	2	1
2	6	3	9	10

x loops up to width

Y loops up to height

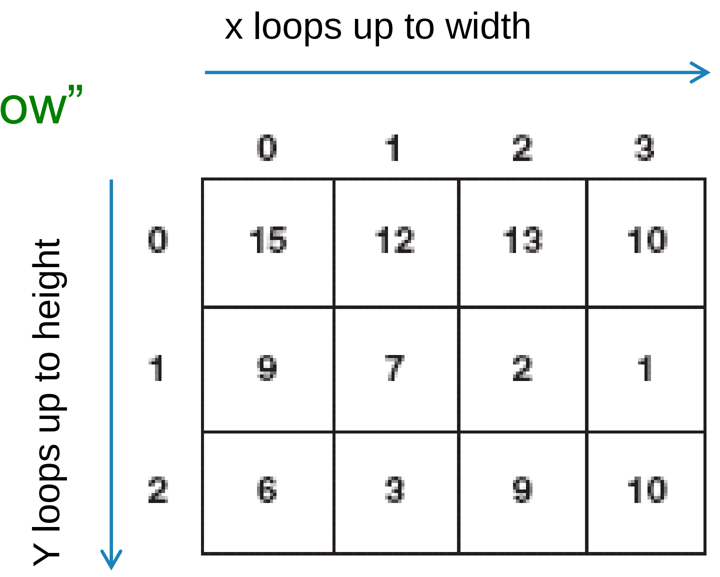
Pt (0, 0)

How to access each pixel

We’ll have to use *nested loops*:

One to walk the width, the other to walk the height.

```
for each y position:  ← “for each row”
  for each x position: ← “for each spot in the row”
        do something to the pixel at (x, y)
```



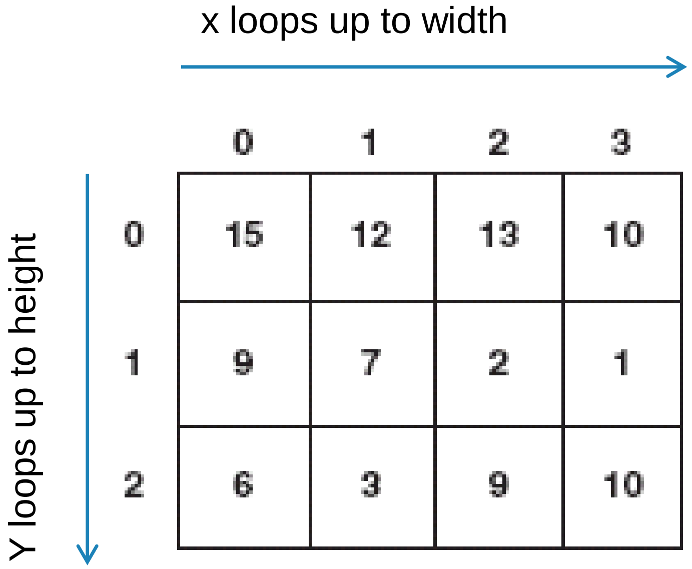
x loops up to width

Y loops up to height

	0	1	2	3
0	15	12	13	10
1	9	7	2	1
2	6	3	9	10

Example on White Board

```
for each y position:  ← “for each row”
  for each x position: ← “for each space in row”
        mark pixel (x,y)
```



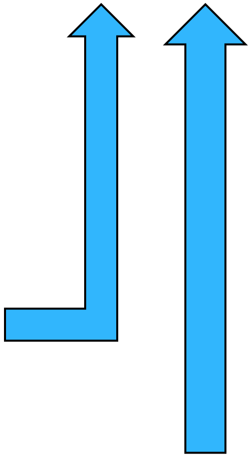
x loops up to width

Y loops up to height

	0	1	2	3
0	15	12	13	10
1	9	7	2	1
2	6	3	9	10

Using ranges

```
for i in range(0, 10):
        print i
```



Start (inclusive)

End (exclusive)

Breaking it down:

1. i is set to 0
2. print i
3. i is then set to 1
4. print I
5. etc…
6. When i is 10, stop

Introducing the function range

- Range returns a sequence between its first two inputs

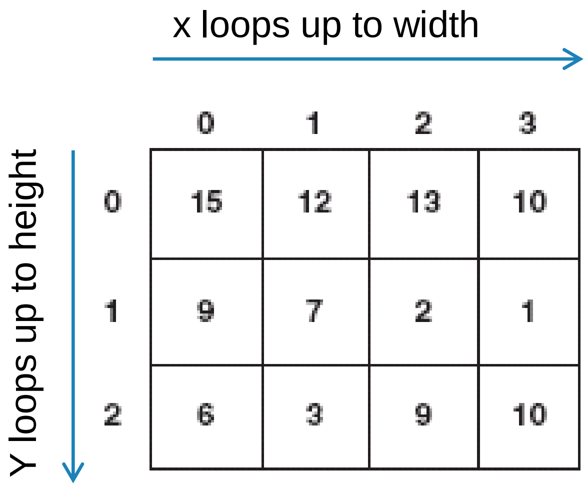
```
>>> print range(0,4)
[0, 1, 2, 3]
>>> print range(-1,3)
[-1, 0, 1, 2]
>>> print range(3)
[0, 1, 2]
```

Using ranges

```
for each y position:  ← "for each row"
  for each x position: ← "for each spot in the row"
      do something to the pixel at (x, y)
```

VS

```
for y in range(0, height):
  for x in range(0, width):
      do something to the pixel at (x, y)
```



	0	1	2	3
0	15	12	13	10
1	9	7	2	1
2	6	3	9	10

Using Ranges

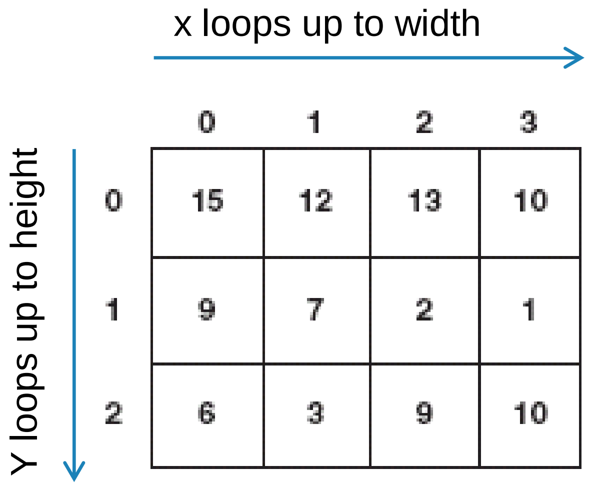
```
for y in range(0, getHeight(picture)):
  for x in range(0, getWidth(picture)):
     pixel = getPixel(picture, x, y)
     setRed(pixel, 5)
```

VS

```
for pixel in getPixels(picture):
    pixel = getPixel(picture, x, y)
    setRed(pixel, 5)
```

Double for loop + Ranges

```
for y in range(1, 3):
  for x in range(1, 4):
       pixel = getPixel(picture, x, y)
       setRed(pixel, 5)
```



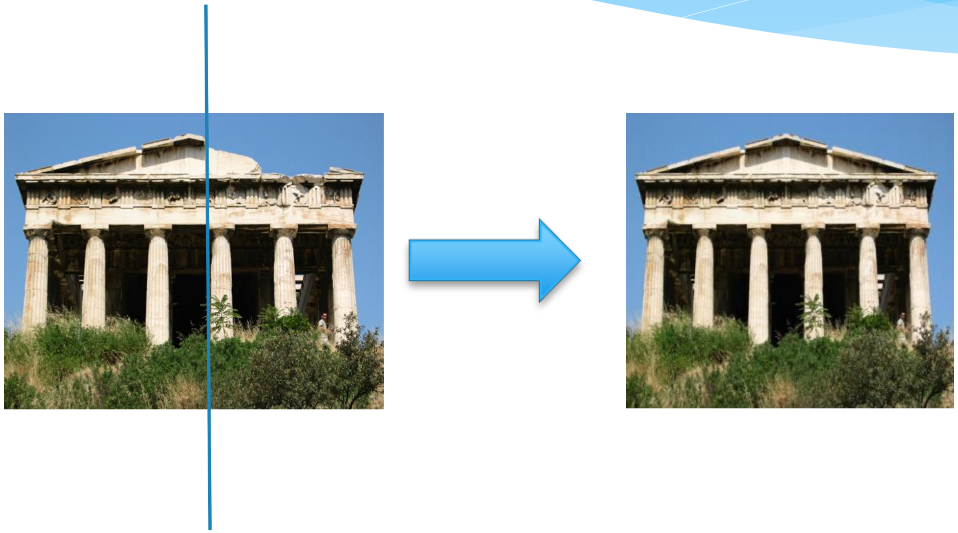
	0	1	2	3
0	15	12	13	10
1	9	7	2	1
2	6	3	9	10

Posterizing with Ranges

```
def posterize(picture):
    for y in range(70,168):
        for x in range(56,190):
            get the red value of the pixel and call it **redValue**
            if **redValue** is less than 64:
                set red of pixel to be 31
            if **redValue** is between 64 and 128:
                set red of pixel to be 95
```

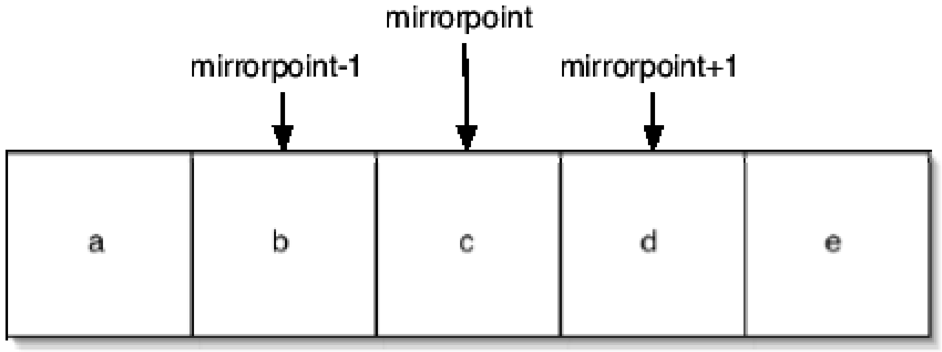
What will the Posterizing function do now?

Mirroring Overview



- For each pixel in the range covering the left half
  - Get distance from the center line
  - **currentColor** = Get color of the pixel
  - Set pixel of same distance from center on right side to the **currentColor**

Mirroring Overview: Example on board



- For each pixel in the range covering the left half
  - Get distance from the center line
  - **currentColor** = Get color of the pixel
  - Set pixel of same distance from center on right side to the **currentColor**

Complete the rest of the lab!

If you have extra time, move on to the extra exercises or experiment further with results from exercises #1 to #4.

WE WILL **STOP 10 MINUTES EARLY** TO SUBMIT YOUR WORK