Params, Returns, Odds and Ends

Looking at the big picture (SimpleImage?) today!

Stanford | ENGINEERING
Computer Science

Housekeeping

- Assignment 1 is due **tonight** (grace period extends to tomorrow night)

- Assignment 2 is released! It is due next Friday, July 14th

- Ecy and Frankie's OH will be in Gates 307, not 303

- The time for midterm/final conflict scheduling is over! If you haven't reached out, you must take the midterm and final during the scheduled windows!

Today

- **Quick recap: Where we are with Python**
  - **What Python do we know?**
  - **How do we run our Python programs?**

- Introducing: Parameters and returns
  - Solidify decomposition
  - Learn about how functions take in information
  - Learn about the `return` statement

Recap: Where we are

- The things we've learned so far can be broken down into:
  - Code we know how to **write**
  - Code we know how to **run**
  - (we are a little bit better at writing code than running it right now)

Code we know how to write

```
while #condition:
    # code that loops
# code that doesn't loop
```

Code we know how to write

```
while #condition:
    # code that loops
```

```
if #condition:
    # code that runs if condition is True
```

Code we know how to write

```
while #condition:
    # code that loops
```

```
if #condition:
    # code that runs if condition is True
```

```
if #condition:
    # block 1 runs if condition is True
else:
    # block 2 runs if condition is False
```

Code we know how to write

```
while #condition:
    # code that loops
```

```
if #condition:
    # code that runs if condition is True
```

```
if #condition:
    # block 1 runs if condition is True
else:
    # block 2 runs if condition is False
```

```
for var_name in range(start_num, end_num):
   # code that loops
```

Which loop?

If we don't know the **number** of times to loop, but we know how to check if we should stop:

```
while #condition:
    # code that loops
```

If we know the number of times we need to loop:

```
for var_name in range(start_num, end_num):
   # code that loops
```

Bonus with for loop: **`var_name`** tells you which loop iteration you are on!

The classic image loop

Do inner loop (a loop through one row) for every row

```
# row by row
for y in range(0, image.height):
    for x in range(0, image.width):
        pixel = image.get_pixel(x, y)
```

Do inner loop (a loop through one col) for every col

```
# col by col
for x in range(0, image.width):
    for y in range(0, image.height):
        pixel = image.get_pixel(x, y)
```

Variables

```
color = 'red' # default color to 'red'
if not bit.get_color() == None:
   # conditionally change value of color
   color = bit.get_color()
```

Variables store a value, and can change the value they store

Variables

```
color = 'red' # default color to 'red'
if not bit.get_color() == None:
   # conditionally change value of color
   color = bit.get_color()
```

Variables store a value, and can change the value they store

```
for x in range(0, image.width):
   pixel = image.get_pixel(x, 0)
```

Variables

```
color = 'red' # default color to 'red'
if not bit.get_color() == None:
   # conditionally change value of color
   color = bit.get_color()
```

Variables store a value, and can change the value they store

```
for x in range(0, image.width):
   pixel = image.get_pixel(x, 0)
```

- x is a variable whose value changes with each loop iteration
- pixel is a variable that we change every time we loop
- 0 is just a value, not a variable

Stanford | ENGINEERING
Computer Science

Running Code

- The experimental server: all you need to do is open a problem, write some code and hit Run!
  - Good for practice problems
  - Get used to Python syntax
  - Bit lives in the experimental server :)

Running Code

- The experimental server: all you need to do is open a problem, write some code and hit Run!
  - Good for practice problems
  - Get used to Python syntax
  - Bit lives in the experimental server :)

- Pycharm: you need to download Pycharm, also download some Python starter code, then open that folder in Pycharm
  - Can write and run any code you want
  - Make big programs!

Pycharm Step by Step

1. Make sure you have Pycharm downloaded according to the handout (linked on website)

2. Download some starter code (like today's lecture code!)

3. Open that folder in Pycharm

4. Open the Terminal (bottom left)

5. Type in **`python3 filename.py [maybe more info]`** and hit enter to run the code in **`filename.py`**

Pycharm Step by Step

1. Make sure you have Pycharm downloaded according to the handout (linked on website)
2. Download some starter code (like today's lecture code!)
3. Open that folder in Pycharm
4. Open the Terminal (bottom left)
5. Type in **`python3 filename.py [maybe more info]`** and hit enter to run the code in **`filename.py`**

   - The "maybe more info" is usually something like a filename or number

   - We will learn more about this later, for now we will always tell you what to put into Terminal

Pycharm code demo

rgb-ify

`and` keyword for conditions

- A lot like the not keyword - use it (and even combine with not) to build more complex conditions
- It means what you think it does!

```
if #condition1 and #condition1:
    # code that runs only if
    # condition1 and condition2 are True
```

`and` keyword for conditions...also `or`

- A lot like the not keyword - use it (and even combine with not) to build more complex conditions
- It means what you think it does!

```
if #condition1 and #condition2:
    # code that runs only if
    # condition1 and condition2 are True
```

```
if #condition1 or #condition2:
    # code that runs if condition1 is True
    # or if condition2 is True, or both!
```

Revisiting Decomp

We've seen how to define helper functions for Bit:

```
def helper_function(bit):
    # must "take in" bit

def main_bit_problem(filename):
    # required first line
    bit = Bit(filename)
    helper_function(bit) # must "pass in" bit
```

Stanford | ENGINEERING
Computer Science

A closer look at parameters

`helper_function` "takes" in **`bit`** as a parameter. This is how **`main_bit_problem`** can communicate with **`helper_function`**

```
def helper_function(bit):
    # must "take in" bit

def main_bit_problem(filename):
    # required first line
    bit = Bit(filename)
    helper_function(bit) # must "pass in" bit
```

Decomposing RGB-ify

```
def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            if pixel.red > pixel.green and pixel.red > pixel.blue:
                pixel.red = 255
                pixel.green = 0
                pixel.blue = 0
            if pixel.green > pixel.red and pixel.green> pixel.blue:
                pixel.red = 255
                pixel.green = 0
                pixel.blue = 0 . . .
```

Decomposing RGB-ify

```
def change_red(???):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
```

Decomposing RGB-ify

```
def change_red(any_pixel):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
```

Decomposing RGB-ify

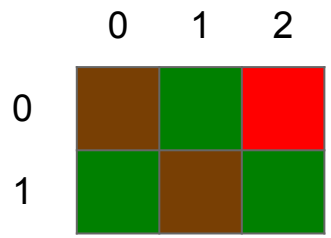
```
def change_red(any_pixel):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
            change_red(pixel)
```

Decomposing RGB-ify

```
def change_red(any_pixel):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        any_pixel.red = 255
    else:
        any_pixel.red = 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
            change_red(pixel)
```

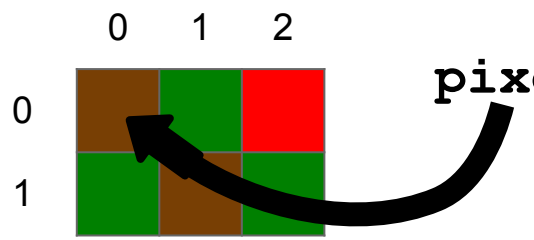
```
def change_red(any_pixel):
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        any_pixel.red = 255
    else:
        any_pixel.red = 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # change_red(pixel) does exactly this
            any_pixel = pixel
            if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
                any_pixel.red = 255
            else:
                any_pixel.red = 0
```

```
def change_red(any_pixel):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        any_pixel.red = 255
    else:
        any_pixel.red = 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
            change_red(pixel)
```



Run `rgb_ify('apple_bush.jpg')`

```
def change_red(any_pixel):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        any_pixel.red = 255
    else:
        any_pixel.red = 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
            change_red(pixel)
```

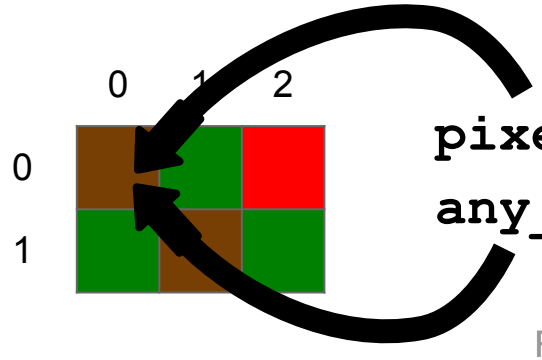


x = 0, y = 0

pixel = RGB(115,55,18) at (0, 0)

Stanford | ENGINEERING
Computer Science

```
def change_red(any_pixel):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        any_pixel.red = 255
    else:
        any_pixel.red = 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
            change_red(pixel)
```



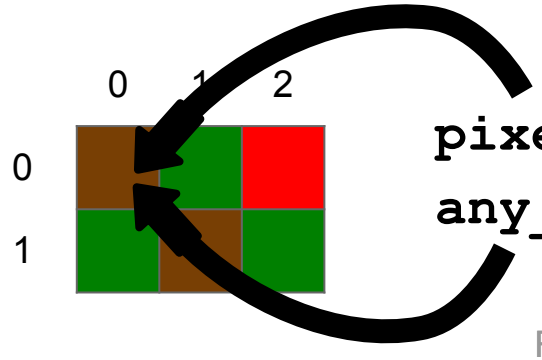
x = 0, y = 0

pixel = RGB(115, 55, 18) at (0, 0)

any_pixel = RGB(115, 55, 18) at (0, 0)

Stanford | ENGINEERING
Computer Science

```
def change_red(any_pixel):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        any_pixel.red = 255
    else:
        any_pixel.red = 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
            change_red(pixel)
```

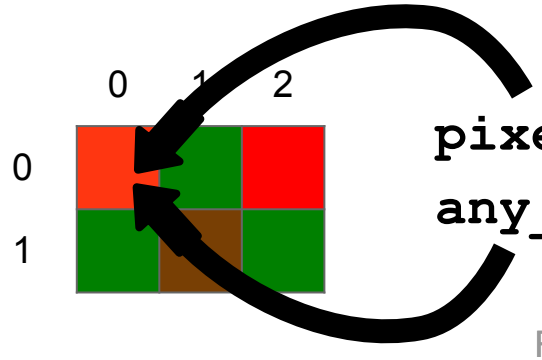


x = 0, y = 0

pixel = RGB(115, 55, 18) at (0, 0)

any_pixel = RGB(115, 55, 18) at (0, 0)

```
def change_red(any_pixel):
    # given a pixel, change its red value to be 255
    # if it is the dominant color, otherwise 0
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        any_pixel.red = 255
    else:
        any_pixel.red = 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use decomp here to change pixel
            change_red(pixel)
```



x = 0, y = 0

pixel = RGB(255, 55, 18) at (0, 0)

any_pixel = RGB(255, 55, 18) at (0, 0)

Modify rgb_ify

Return values

- In `change_red`, we modified the pixel, and that how we communicated back to the calling function what we did
- The **`return`** keyword is another way to do this
- We can return values from helper function and collect those values in the calling function

Decomposing RGB-ify with return values

```
def get_red(any_pixel):
    # given a pixel, determine what its red value should be
    # (either 0 or 255) and return that value


def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # save return value of get_red in pixel.red
```

Decomposing RGB-ify with return values

```
def get_red(any_pixel):
    # given a pixel, determine what its red value should be
    # (either 0 or 255) and return that value


def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # save return value of get_red in pixel.red
            pixel.red = get_red(pixel)
```

Decomposing RGB-ify with return values

```
def get_red(any_pixel):
    # given a pixel, determine what its red value should be
    # (either 0 or 255) and return that value
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        return 255
    else:
        return 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # save return value of get_red in pixel.red
            pixel.red = get_red(pixel)
```

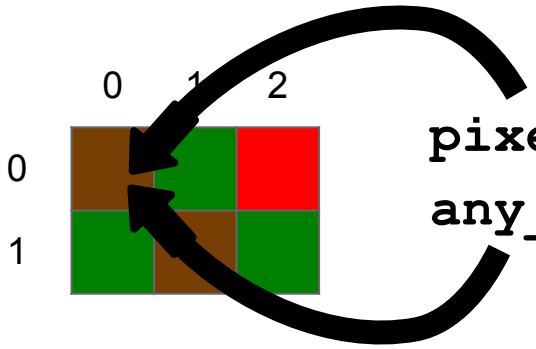
```
def get_red(any_pixel):
    if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
        return 255
    else:
        return 0

def rgb_ify(filename):
    image = SimpleImage(filename)
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # pixel.red = get_red(pixel) does exactly this
            any_pixel = pixel
            if any_pixel.red > any_pixel.green and any_pixel.red > pixel.blue:
                return_value = 255
            else:
                return_value = 0
            pixel.red = return_value
```

Stanford | ENGINEERING
Computer Science

Modify rgb_ify again!

Modifying vs moving

A "copy" of a pixel (and most "large" variable types) is really a copy of a pointer to the original item

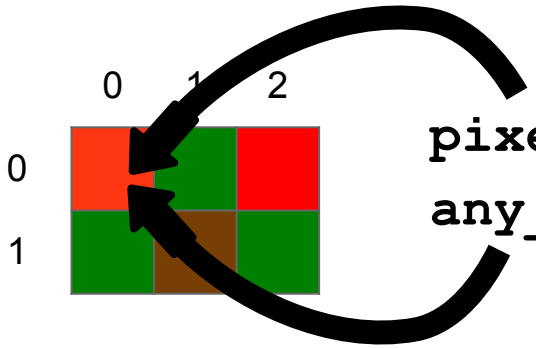



```
x = 0, y = 0
pixel = RGB(115,55,18) at (0, 0)
any_pixel = RGB(115,55,18) at (0, 0)
```

Modifying vs moving

A "copy" of a pixel (and most "large" variable types) is really a copy of a pointer to the original item

Modifying an attribute of the copy modifies that attribute in the original - that change will persist



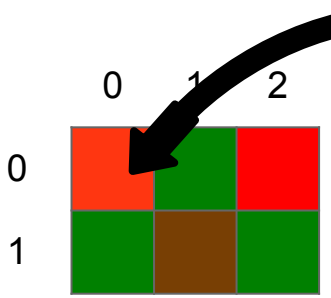

```
x = 0, y = 0
pixel = RGB(255, 55, 18) at (0, 0)
any_pixel = RGB(255, 55, 18) at (0, 0)
any_pixel.red = 255
```

Modifying vs moving

A "copy" of a pixel (and most "large" variable types) is really a copy of a pointer to the original item

Modifying an attribute of the copy modifies that attribute in the original - that change will persist

Moving the copy to point to a different pixel will not change where the original points - this change won't persist




```
x = 0, y = 0
pixel = RGB(255, 55, 18) at (0, 0)
any_pixel = RGB(255, 55, 18) at (0, 0)
any_pixel = some_other_pixel
```

copy_last_pixel

- Write the function copy_last_pixel(filename), takes the image specified by filename and changes every pixel to look like the bottom-right (last) pixel
- Decompose copying one pixel

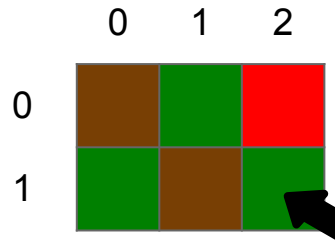
```
def copy_pixel(???):
    # given two pixels, make the first pixel
    # the same as the second pixel

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    # first: save last pixel
```

```
def copy_pixel(???):
    # given two pixels, make the first pixel
    # the same as the second pixel

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)
```

```
def copy_pixel(???):
    # given two pixels, make the first pixel
    # the same as the second pixel

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(???)
```

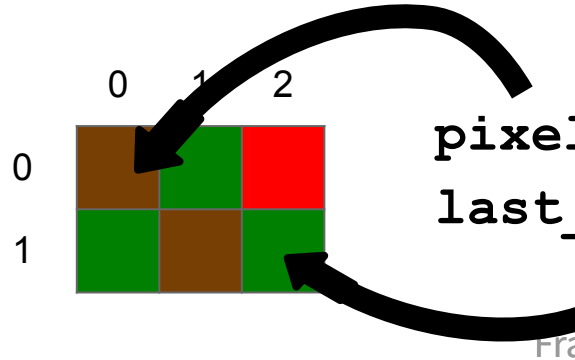
```
def copy_pixel(pixel_1, pixel_2):
    # given two pixels, make the first pixel
    # the same as the second pixel

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```

```
def copy_pixel(pixel_1, pixel_2):
    # given two pixels, make the first pixel
    # the same as the second pixel
    pixel_1 = pixel_2 # first attempt

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```



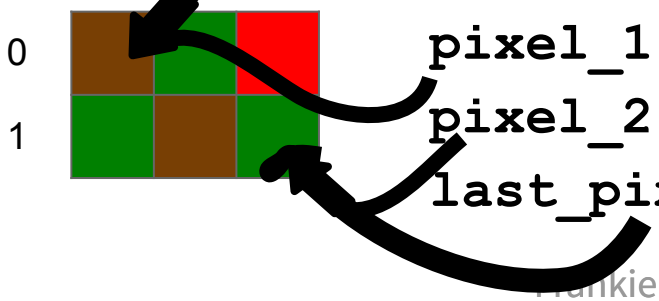
last_pixel = RGB(0,255,0) at (2, 2)

```
def copy_pixel(pixel_1, pixel_2):
    # given two pixels, make the first pixel
    # the same as the second pixel
    pixel_1 = pixel_2 # first attempt

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```



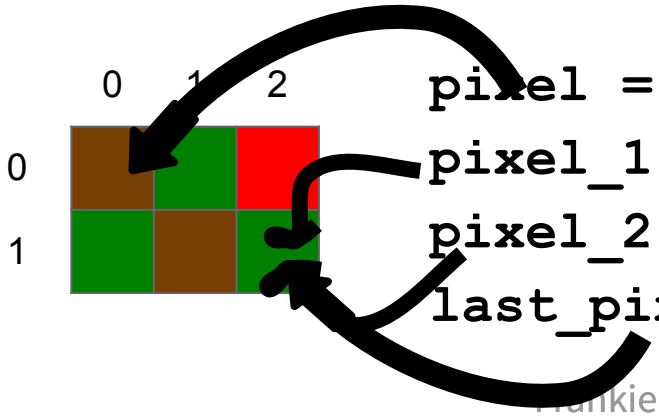

```
x = 0, y = 0
pixel = RGB(255,55,18) at (0, 0)
last_pixel = RGB(0,255,0) at (2, 2)
```

```
def copy_pixel(pixel_1, pixel_2):
    # given two pixels, make the first pixel
    # the same as the second pixel
    pixel_1 = pixel_2 # first attempt

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```



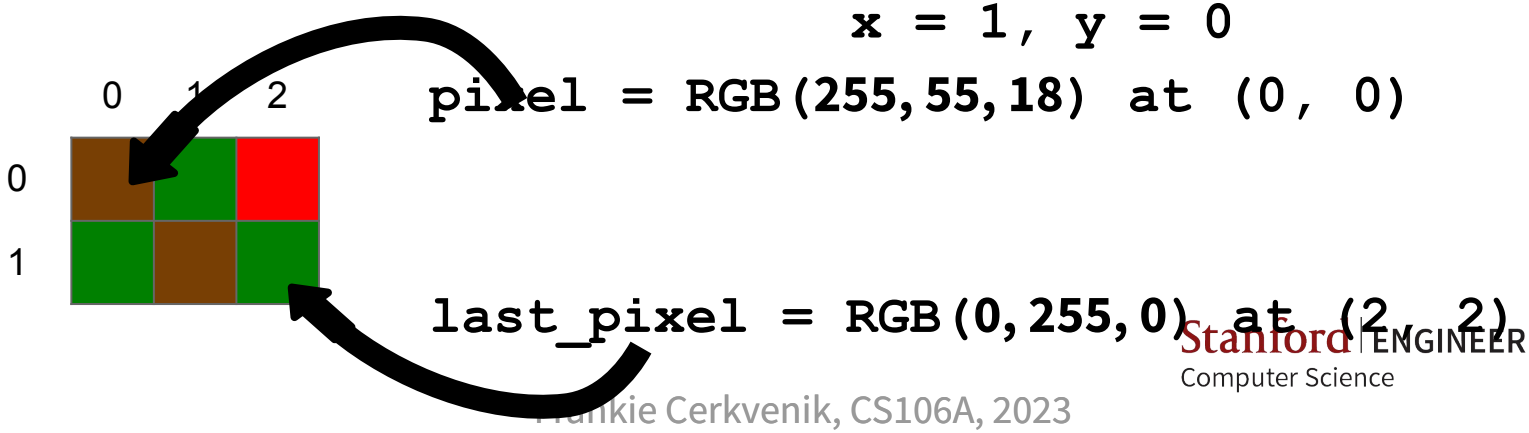

```
                x = 0, y = 0
pixel = RGB(255,55,18) at (0, 0)
pixel_1 = RGB(255,55,18) at (0, 0)
pixel_2 = RGB(0,255,0) at (2, 2)
last_pixel = RGB(0,255,0) at (2, 2)
```

```
def copy_pixel(pixel_1, pixel_2):
    # given two pixels, make the first pixel
    # the same as the second pixel
    pixel_1 = pixel_2 # first attempt

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```

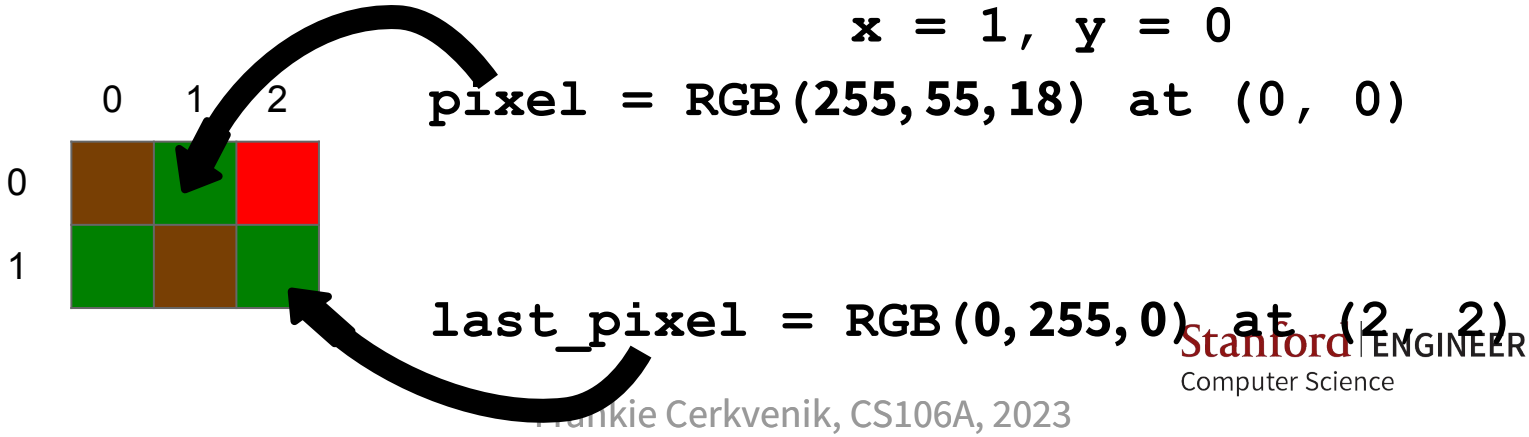



```
                x = 0, y = 0
pixel = RGB(255,55,18) at (0, 0)
pixel_1 = RGB(255,55,18) at (0, 0)
pixel_2 = RGB(0,255,0) at (2, 2)
last_pixel = RGB(0,255,0) at (2, 2)
```

```
def copy_pixel(pixel_1, pixel_2):
    # given two pixels, make the first pixel
    # the same as the second pixel
    pixel_1 = pixel_2 # first attempt

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```



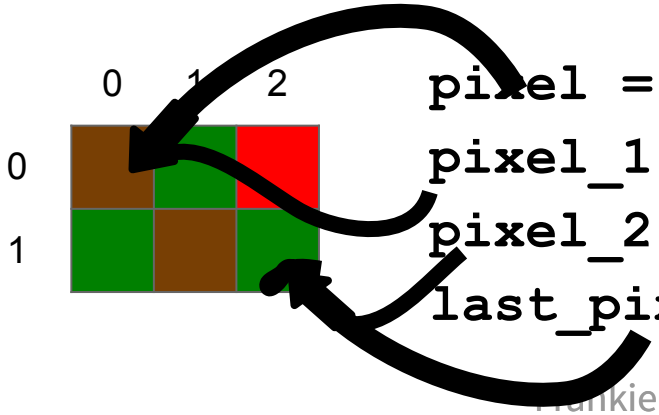
```
def copy_pixel(pixel_1, pixel_2):
    # given two pixels, make the first pixel
    # the same as the second pixel
    pixel_1 = pixel_2 # first attempt

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```



Modify don't move in helpers!

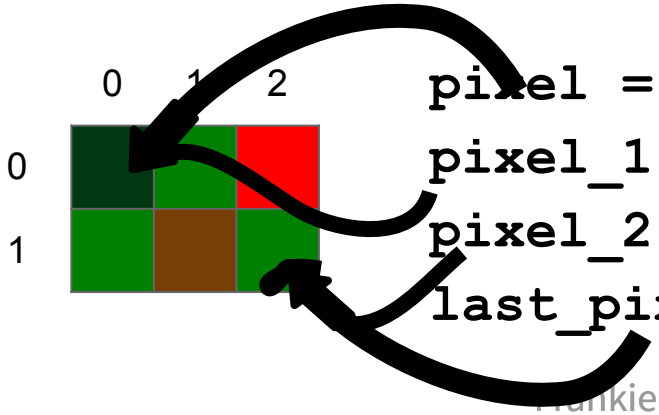
```
def copy_pixel(pixel_1, pixel_2):
    #fix: modify not move
    pixel_1.red = pixel_2.red
    pixel_1.green = pixel_2.green
    pixel_1.blue = pixel_2.blue

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```

```
def copy_pixel(pixel_1, pixel_2):
    pixel_1.red = pixel_2.red
    pixel_1.green = pixel_2.green
    pixel_1.blue = pixel_2.blue

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```



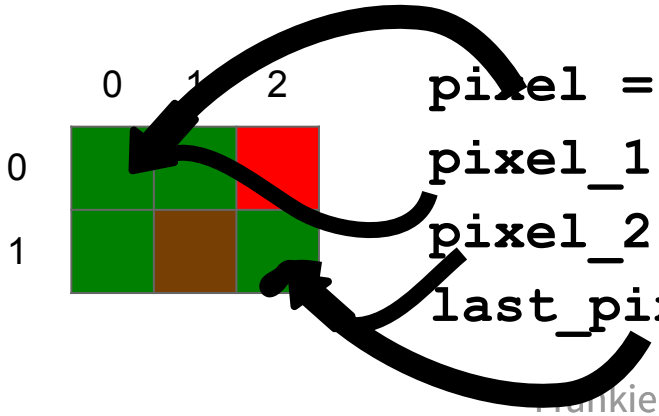

```
                    x = 0, y = 0
pixel = RGB(255,55,18) at (0, 0)
pixel_1 = RGB(255,55,18) at (0, 0)
pixel_2 = RGB(0,255,0) at (2, 2)
last_pixel = RGB(0,255,0) at (2, 2)
```

```
def copy_pixel(pixel_1, pixel_2):
    pixel_1.red = pixel_2.red
    pixel_1.green = pixel_2.green
    pixel_1.blue = pixel_2.blue

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```




```
                    x = 0, y = 0
pixel = RGB(0,55,18) at (0, 0)
pixel_1 = RGB(0,55,18) at (0, 0)
pixel_2 = RGB(0,255,0) at (2, 2)
last_pixel = RGB(0,255,0) at (2, 2)
```

```
def copy_pixel(pixel_1, pixel_2):
    pixel_1.red = pixel_2.red
    pixel_1.green = pixel_2.green
    pixel_1.blue = pixel_2.blue

def copy_last_pixel(filename):
    image = SimpleImage(filename)
    last_pixel = image.get_pixel(image.width - 1, image.height - 1)

    # loop through every pixel and copy the last pixel to it
    for y in range(0, image.height):
        for x in range(0, image.width):
            pixel = image.get_pixel(x, y)
            # use our decomped helper!
            copy_pixel(pixel, last_pixel)
```




```
                    x = 0, y = 0
pixel = RGB(0,55,18) at (0, 0)
pixel_1 = RGB(0,55,18) at (0, 0)
pixel_2 = RGB(0,255,0) at (2, 2)
last_pixel = RGB(0,255,0) at (2, 2)
```

Recap: “Changing” parameters in helpers

- You may **modify attributes** of parameters in helper functions, and those changes will persist

- You may not “move” or set parameters equal to other values - those changes will not persist

```
parameter.attribute = something
```
✅✅✅

```
parameter = something
```
🚫🚫🚫

(If time) Experimental server practice: darker_left

- Given parameters filename and left, create an image from filename with the left% of the left side of the image darker
- We can make a pixel darker by halving the RGB values in the pixel

```
darker_left('poppy.jpg', 50)
```

```
darker_left('poppy.jpg', 20)
```

Stanford | ENGINEERING
Computer Science

Darker Left

Moving away from images

- Most of the variables we have worked with have been pixels and images

- They can also be numbers!

- They can also be boolean values (True/False)

- They can also be any other type of value (stay tuned!)

Recall: Variables as numbers

```
def variable_num_example(filename):
  x = 3 # numbers without decimal are "ints"
  y = 8.0 # numbers with decimal are "floats"

  # math is math
  sum = x + y # sum is 11
  prod = x * y # prod is 24
  diff = x - y # diff is -5

  # regular division
  quotient = y / x # quotient is 2.6666

  # int division
  int_quotioent = y // x
  # int_quotient is 2, // truncates decimal
```

Introducing: The `print` function

- The **`print`** function takes in (as a parameter) something to print
- It displays what you passed in on the Terminal!
- Works best with simpler types, like:
  - Ints
  - Floats
  - Anything in quotes, like **`"hello"`** (we call these strings)
- **Printing is not the same as returning**

Introducing: The `print` function

- The **`print`** function takes in (as a parameter) something to print
- It displays what you passed in on the Terminal!
- Works best with simpler types, like:
  - Ints
  - Floats
  - Anything in quotes, like **`"hello"`** (we call these strings)
- **Printing is not the same as returning**

```
def variable_num_example(filename):
  x = 3
  print(x) # displays 3
  print("hello world") # displays hello world
```

factorial_average

- Let's write a function, **`factorial_average(num1, num2)`**, which takes in two numbers, prints their factorials, and then prints the average of their factorials.

- Decompose "calculate the factorial of a number" and "calculate the average of a number" into two helper functions

- Side note: don't worry if this goes fast, we will revisit it on Tuesday!

factorial_avg

(in pycharm)

“Changing” parameters

Lets try a “change” version of **`compute_avg`**

```
def change_num1_to_avg(num1, num2):
    num1 = (num1 + num2) / 2
```

“Changing” parameters

Lets try a “change” version of **`compute_avg`**

```
def change_num1_to_avg(num1, num2):
  num1 = (num1 + num2) / 2

def use_change_avg():
  x = 1
  y = 3
  print(x)
  change_num1_to_avg(x, y)
  print(x)
```

“Changing” parameters

Lets try a “change” version of **`compute_avg`**

```
def change_num1_to_avg(num1, num2):
    num1 = (num1 + num2) / 2

def use_change_avg():
    x = 1
    y = 3
    print(x)
    change_num1_to_avg(x, y)
    print(x)
```

Variables:

`x = 1`

`y = 3`

Console prints:

1

Stanford | ENGINEERING
Computer Science

“Changing” parameters

Lets try a “change” version of **`compute_avg`**

```
def change_num1_to_avg(num1, num2):
    num1 = (num1 + num2) / 2

def use_change_avg():
    x = 1
    y = 3
    print(x)
    change_num1_to_avg(x, y)
    print(x)
```

Variables:

`x = 1` `num1 = 1`

`y = 3` `num2 = 3`

Stanford | ENGINEERING
Computer Science

“Changing” parameters

Lets try a “change” version of **`compute_avg`**

```
def change_num1_to_avg(num1, num2):
    num1 = (num1 + num2) / 2

def use_change_avg():
    x = 1
    y = 3
    print(x)
    change_num1_to_avg(x, y)
    print(x)
```

Variables:

`x = 1` `num1 = 1.5`

`y = 3` `num2 = 3`

Stanford | ENGINEERING
Computer Science

“Changing” parameters

Lets try a “change” version of **`compute_avg`**

```
def change_num1_to_avg(num1, num2):
    num1 = (num1 + num2) / 2

def use_change_avg():
    x = 1
    y = 3
    print(x)
    change_num1_to_avg(x, y)
    print(x)
```

Variables:

`x = 1`

`y = 3`

~~`num1 = 1.5`~~

~~`num2 = 3`~~

Console prints:

???

Stanford | ENGINEERING
Computer Science

“Changing” parameters

🚫🚫The change does not persist 🚫🚫

```
def change_num1_to_avg(num1, num2):
    num1 = (num1 + num2) / 2

def use_change_avg():
    x = 1
    y = 3
    print(x)
    change_num1_to_avg(x, y)
    print(x)
```

Variables:

`x = 1`

`y = 3`

~~`num1 = 1.5`~~

~~`num2 = 3`~~

Console prints:

1

Stanford | ENGINEERING
Computer Science

“Changing” parameters

The change happens to num1, not x

```
def change_num1_to_avg(num1, num2):
    num1 = (num1 + num2) / 2
    print(num1)

def use_change_avg():
    x = 1
    y = 3
    print(x)
    change_num1_to_avg(x, y)
    print(x)
```

Variables:

`x = 1`

`y = 3`

`num1 = 1.5`

`num2 = 3`

Console prints:

1.5

Stanford | ENGINEERING
Computer Science

Why? Copies!

A copy of an int is literally a different item

Changing the copy (`num1`) won't change the original (`x`)

Variables:

```
x = 1          num1 = 1.5
y = 3          num2 = 3
```

Recap

- Parameters are how helper functions receive information from the functions that call them

- If you modify an attribute of a parameter, that change will persist to the calling function, but if you move it (**`param = other_thing`**), that change won't persist

- Returns are how callers receive information back from the functions they call

- All the code we've been writing works with numbers too :)

Stanford | ENGINEERING
Computer Science