Images
CS106A, Stanford University
Housekeeping

• Handout: Image Reference Guide
  – We'll be talking through a lot of that today

• Katie Creel will give a guest mini-lecture on the ethics of image manipulation next class
  – There are questions about that on Assignment #3, so you definitely want to pay attention!
More on Lists
Lists as Parameters

• When you pass a list as a parameter you are passing a reference to the actual list
  – It's like getting a URL to the list (*pass-by-reference*)
  – In function, changes to values in list persist after function ends

```python
def add_five(num_list):
    for i in range(len(num_list)):
        num_list[i] += 5

def main():
    values = [5, 6, 7, 8]
    add_five(values)
    print(values)
```

Output: `[10, 11, 12, 13]`
## When Passed as Parameters

<table>
<thead>
<tr>
<th>Types that are &quot;immutable&quot;</th>
<th>Types that are &quot;mutable&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>list</td>
</tr>
<tr>
<td>float</td>
<td>(we'll see more soon)</td>
</tr>
<tr>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

- When you assign a new value to a variable, you are assigning luggage tag (name) to a new value.
- For parameters, the original variable value you passed in is **not** changed when function is done.

- When you are changing the variable **in place**, the luggage tag does not change, but the value inside the luggage does.
- For parameters, it means original variable value you passed in **is** changed when function is done.
More on Lists as Parameters

• But, watch out if you create a new list in a function
  – Creating a new list means you're no longer dealing with list passed in as parameter.
  – It's like the URL you are using is pointing to a different page.
    (You have assigned the luggage tag to a new value in function.)
  – At that point you are no longer changing parameter passed in

```python
def create_new_list(num_list):
    num_list.append(9)
    num_list = [1, 2, 3]

def main():
    values = [5, 6, 7, 8]
    create_new_list(values)
    print(values)

Output [5, 6, 7, 8, 9]
```
list = [10, 20, 30]

• For loop using `range`:
  ```python
  for i in range(len(list)):
      list[i] += 1  # Modifying list in place
  ```

• For-each loop:
  ```python
  for elem in list: # Modifying local variable
      elem += 1      # elem. If elem is immutable
                      # type, not changing list!
  ```

• Often use for loop with `range` when modifying elements of list (when elements are immutable types)

• Often use for-each loop when not modifying elements of list or when elements are mutable types
Putting it all together:
averagescores.py
Learning Goals: Images

1. Understanding how images are represented
2. Learning about the SimpleImage library
3. Writing code that can manipulate images
Images
What is an Image?

• Image made of square pixels
  – Example: flower.png

• Each pixel has x and y coordinates in the image
  – The origin (0, 0) is at the upper-left corner
  – y increases going down, x increases going right

• Each pixel has single color encoded as 3 RGB values
  – R = red; G = green; B = blue
  – Each value represents brightness for that color (red, green, or blue)
  – Can set RGB values to make any color!
Pixels in an Image Close-Up

Pixel (1, 0):
- red: 6
- green: 250
- blue: 7
  (i.e. shade of green)

Pixel (4, 2):
- red: 241
- green: 252
- blue: 23

Pixel (2, 3):
- red: 247
- green: 250
- blue: 237
Working with Images: Pillow and the SimpleImage library
Installing Pillow

• Pillow is a version of the Python Imaging Library (PIL)
  – Nick Parlante built SimpleImage library using Pillow
  – You'll be using SimpleImage in this class
  – So, you need to install Pillow first

• To install Pillow, open PyCharm Terminal tab and type (note the capital P in Pillow):
  – On a PC:  `py -m pip install Pillow`
  – On a Mac:  `python3 -m pip install Pillow`
  – Will see something like:

    ...bunch of stuff...

    Successfully installed Pillow-9.1.0

• Handout: Image Reference Guide contains more information
Using SimpleImage Library

- In folders for assignment or lecture on images, there is a file `simpleimage.py`  
  - This is the SimpleImage library

- To use the SimpleImage library in your code, include at the top of your program file:

  ```python
  from simpleimage import SimpleImage
  ```

- This is importing the SimpleImage module, so that it is accessible in the code you write
  - Similar to when you used `import random` to use random number generator library
Functions in SimpleImage Library

• Create a SimpleImage object by reading an image from file (jpg, png, gif, etc.) and store it in a variable.
  – Note: each SimpleImage object is made up of Pixel objects
    
    ```
    my_image = SimpleImage(filename)
    ```

• Show the image on your computer.
  ```
  my_image.show()
  ```

• We can manipulate an image by changing its pixels
• We can also create new images and set its pixels
Accessing Pixels in an Image

• We can use a "for-each" loop to access pixel in an image

• Recall basic for loop (using range):

```python
for i in range(num):
    # i will go from 0 to num - 1
do_something()
```

• For-each loop:

```python
for item in collection:
    # Do something with item
```

• For-each loop with image:

```python
image = SimpleImage("flower.jpg")
for pixel in image:
    # Do something with pixel
```
image = SimpleImage("flower.jpg")
for pixel in image:
    # Body of loop
    # Do something with pixel

- Like variable i in for loop using range(), pixel is a variable that gets updated with each loop iteration.
- pixel gets assigned to each pixel object in the image in turn.

This code gets repeated once for each pixel in image.
Properties of Images and Pixels

- Each SimpleImage image has properties you can access:
  - Can get the width and height of image (values are in pixels)
    `image.width, image.height`

- Each pixel in an image also has properties:
  - Can get x, y coordinates of a pixel in an image
    `pixel.x, pixel.y`
  - Can get RGB values of a pixel
    `pixel.red, pixel.green, pixel.blue`
    • These are just integers between 0 and 255
    • Higher R, G, or B values means more of that color in pixel
  - Pixels are mutable objects!
  - Can set pixel RGB values in an image to change it!
Example: A Darker Image

```python
def darker(image):
    
    """
    Makes image passed in darker by halving red, green, blue values. Note: changes in image persist after function ends.
    """

    # Demonstrate looping over all the pixels of an image, # changing each pixel to be half its original intensity.
    for pixel in image:
        pixel.red = pixel.red // 2
        pixel.green = pixel.green // 2
        pixel.blue = pixel.blue // 2

def main():
    flower = SimpleImage('images/flower.png')
    darker(flower)
    flower.show()
```

Image objects are mutable (like lists). If you change one in a function, the changes persist after function ends.
def red_channel(filename):
    """
    Reads image from file specified by filename.
    Changes the image as follows:
    For every pixel, set green and blue values to 0
    yielding the red channel.
    Return the changed image.
    """
    image = SimpleImage(filename)
    for pixel in image:
        pixel.green = 0
        pixel.blue = 0
    return image
def compute_luminosity(red, green, blue):
    """
    Calculates luminosity of a pixel using NTSC formula.
    """
    return (0.299 * red) + (0.587 * green) + (0.114 * blue)

def grayscale(filename):
    """
    Read image from file specified by filename. Change image to grayscale using the NTSC luminosity formula and return it.
    """
    image = SimpleImage(filename)
    for pixel in image:
        lum = compute_luminosity(pixel.red, pixel.green, pixel.blue)
        pixel.red = lum
        pixel.green = lum
        pixel.blue = lum
    return image
Let's take it out for a spin!
imageexamples.py
Greenscreening
What is Greenscreening?

• Like the movies (and Zoom backgrounds)
  – Have original image with areas that are "sufficiently green."
  – Replace "green" pixels with pixels from corresponding x, y locations in another image
What is Greenscreening?

- Like the movies (and Zoom backgrounds)
  - Have original image with areas that are "sufficiently green."
  - Replace "green" pixels with pixels from corresponding x, y locations in another image

INTENSITY_THRESHOLD = 1.6

```python
def greenscreen(main_filename, back_filename):
    image = SimpleImage(main_filename)
    back = SimpleImage(back_filename)
```
What is Greenscreening?

- Like the movies (and Zoom backgrounds)
  - Have original image with areas that are "sufficiently green."
  - Replace "green" pixels with pixels from corresponding x, y locations in another image

```
INTENSITY_THRESHOLD = 1.6

def greenscreen(main_filename, back_filename):
    image = SimpleImage(main_filename)
    back = SimpleImage(back_filename)
    for pixel in image:
```
What is Greenscreening?

- Like the movies (and Zoom backgrounds)
  - Have original image with areas that are "sufficiently green."
  - Replace "green" pixels with pixels from corresponding x, y locations in another image

\[
\text{INTENSITY\_THRESHOLD} = 1.6
\]

```python
def greenscreen(main_filename, back_filename):
    image = SimpleImage(main_filename)
    back = SimpleImage(back_filename)
    for pixel in image:
        average = (pixel.red + pixel.green + pixel.blue) // 3
        # See if this pixel is "sufficiently" green
        if pixel.green >= average * INTENSITY\_THRESHOLD:
```
What is Greenscreening?

• Like the movies (and Zoom backgrounds)
  – Have original image with areas that are "sufficiently green."
  – Replace "green" pixels with pixels from corresponding x, y locations in another image

INTENSITY_THRESHOLD = 1.6

```python
def greenscreen(main_filename, back_filename):
    image = SimpleImage(main_filename)
    back = SimpleImage(back_filename)
    for pixel in image:
        average = (pixel.red + pixel.green + pixel.blue) // 3
        # See if this pixel is "sufficiently" green
        if pixel.green >= average * INTENSITY_THRESHOLD:
            # If so, overwrite pixel in original image with corresponding pixel from the back image.
            x = pixel.x
            y = pixel.y
            image.set_pixel(x, y, back.get_pixel(x, y))
    return image
```
Let's try it!

(But using red instead of green)
Learning Goals

1. Understanding how images are represented
2. Learning about the SimpleImage library
3. Writing code that can manipulate images