

Image Reference Guide

Install Pillow

The SimpleImage module we have seen in class uses the "Pillow" library. A library is a body of already written code which you import and use, and in this case the Pillow library contains code to manipulate images. In order for SimpleImage to work properly, you need to install Pillow on your machine. We'll talk about libraries in more detail later in the quarter.

To install Pillow, you should first open a "terminal" window: the easiest way to do this is to use the "Terminal" tab within PyCharm on the lower-left (next to the "Run" and "Python Console" tabs). Type the following command shown in bold below into the Terminal (Note that "Pillow" starts with an uppercase P. On Windows, type "**py**" instead of "**python3**"):

```
$ python3 -m pip install Pillow  
..prints stuff...  
Successfully installed Pillow-6.1.0
```

To test that Pillow is working, type the following command (in bold) into a terminal inside any assignment or lecture example folder that contains the file '**simpleimage.py**' (On windows, type "**py**" instead of "**python3**"):

```
# (inside any assignment or lecture example project that uses  
SimpleImage)  
$ python3 simpleimage.py  
# yellow rectangle with green stripe appears
```

This runs the '**simpleimage.py**' code included in the folder. When run like this, you should see a big yellow rectangle with a green stripe on the right pop up. **If you cannot get Pillow installed successfully, please post on Piazza or go to course staff office hours.**

Note: You only have to install Pillow once. After you have successfully installed it, the library will be available for use in all future assignments and lecture examples.

Basic SimpleImage Functionality

The SimpleImage code used in CS106AP provides you with some basic digital image processing tools.

- **Creating an image from a filename.** This function reads the image into memory from the specified file and returns a SimpleImage object that can be stored in a variable to refer to the image.

```
image = SimpleImage(filename)
```

- **For-each loop over all pixels in the image.** This capability gives you easy access to each of the pixels in the image so that you can apply the same code to all of them.

```
for pixel in image:  
    # Use pixel in here
```

- **Accessing data inside a pixel.** Each pixel has properties that can be accessed with the “dot” syntax. These properties include its RGB components (red, blue, and green), as well as the location of the pixel in its corresponding image. The color properties can be accessed and modified, but the location properties cannot be directly modified.

Color properties: `pixel.red`, `pixel.green`, `pixel.blue`

Location properties: `pixel.x`, `pixel.y`

- **Displaying the image.** Each image can be displayed on your computer using the `.show()` function. Usually, you will write functions that return images and then display the images you want to see inside the `main()` function.

```
image.show()
```

- **Accessing image size.** Each image has two attributes – height and width – that can be directly accessed from the image. This is useful for algorithms that rely on the size of the image in some way.

```
image.width, image.height
```

- **Enforcing valid RGB values.** The RGB values are stored as integer values in the range 0-255. Each pixel has a red value, a green value, and a blue value. RGB values outside the valid range are changed to be an integer between 0-255 in the following manner:

- a. Setting a value to a float like 3.5 is changed to the int 3 internally
- b. Setting a value larger than 255 is changed to 255 internally
- c. Setting a value less than 0 is changed to 0 internally

Advanced SimpleImage Functionality

As you progress into creating your own images or working on more complicated image algorithms, there are some more advanced features of SimpleImage that you can take advantage of.

- **Creating a blank image with a specified height and width.** This is useful for starting off with a “blank canvas” that you can populate in an algorithmically interesting way. Both of the lines of code below are valid ways of making a blank image with a width of 200 pixels and a height of 100 pixels.

```
image = SimpleImage.blank(200, 100)
```

```
image = SimpleImage.blank(width=200, height=100)
```

- **Accessing a pixel by its x,y coordinates.** The top left of the image corresponds to `x=0`, `y=0`. x-coordinates get larger as you move to the right, and y-coordinates get larger as you move down.

```
pixel = image.get_pixel(x, y)
```

Note: **x** and **y** should be integers and **x** must be in the range 0 to **width-1** (**inclusive**) and **y** must be in the range 0 to **height-1** (**inclusive**)

- **Setting the value of a certain pixel in an image by its x,y coordinates.** If you have a pixel and you want to place it directly into an image, you can do so with the following function.

```
image.set_pixel(x, y, pixel)
```

- **Resizing an image to be the same size as another image.** This functions takes in the **target_image** whose size you want to match and modifies **image** so that it has identical dimensions to the target image. This function is especially helpful when working on bluescreen-type problems - it helps you ensure that the sizes of your foreground and background images match up.

```
image.make_as_big_as(target_image)
```

Range Loops with SimpleImage

The "for each" loop described above is the easiest way to loop over all of the pixels in an image. However, sometimes you want to write loops that access pixels by their **x, y** coordinates. The code below demonstrates the canonical nested range loop pattern to access all of the pixels in an image. The outer for loop iterates over the rows (starting with the top row (**y=0**) and moving on to the next row (**y=1**)), and the inner for loop iterates over the columns.

```
def example(filename):  
    image = SimpleImage(filename)  
    for y in range(image.height):  
        for x in range(image.width):  
            pixel = image.get_pixel(x, y)  
            # do something with pixel  
            pixel.red = pixel.red // 2  
    return image
```