Computer Science 1 — CSci 1100
Lab 4 — Images and Modules

This lab explores the use of images and modules. We strongly urge you to reference the Lecture 7 notes from last Thursday's class:

`http://www.cs.rpi.edu/academics/courses/fall16/cs1/lecture_notes/lec07_modules_images.html`

You can also find more details on all image functions at:

`http://pillow.readthedocs.org/en/latest/handbook/tutorial.html`

In this lab, you have two mandatory checkpoints that everyone must complete. For the third checkpoint, we will give you a couple of options. You must complete at least one of these, but you are welcome to try all of them. We provide a module called `panoramio.py` which will be used in one of the possible checkpoint 3 options. This is a good exercise to use an external module as part of your code.

Before you start, we recommend that you verify your pillow installation. Open your Wing IDE and go to your python shell pane. Enter:

```
from PIL import Image
im = Image.new('RGB', (200,200))
```

If you get any errors, it is likely that you did not get `pillow` properly installed during Lab 0. Please visit:

`http://www.cs.rpi.edu/academics/courses/fall16/cs1/python_environment.html`,

search for `conda install pillow`, and follow the directions provided.

Checkpoint 1: Two by two wallpaper

To start this part, first download the `lab04files.zip` file from Piazza. The file contains many images along with the file `panoramio.py` and some additional code to test and verify the panorama module. Unzip this folder in your Dropbox folder for Lab4. For simplicity, make sure that your programs are saved in the same directory as these images before you try to execute it. Otherwise, you will get a `File Not Found` error when trying to open the images.

In this first part of the lab, you are going to create two by two a wallpaper using any four of the images. We recommend images titled `'ca.jpg','im.jpg', 'hk.jpg','bw.jpg'`, but you can use any image you wish. For this lab, you are going to hard code the names of the images into your program.

Open a new file and save it out as `check1.py`.

Next, create a `512x512` blank document. Then, open each image, resize it to size `256x256` and paste them into this new image to form a 2x2 wallpaper.

All functions to do this are illustrated in the examples in the Lecture 7 notes. And, remember, the paste function modifies an image but does not return anything.

When done, call the `show()` function to check that the wallpaper looks correct. You will see that the images are distorted because the original images are not square and the resize does not preserve

the aspect ratio. We will fix this issue in for `Checkpoint 2`.

For your convenience, here are a few image functions that might help you:

```
from PIL import Image    ##must import Image first to be able to use it
im = Image.new(mode, size, color)  # use mode 'RGB', color 'white'
im = im.resize((width,height)) # resize to the given width/height passed as a tuple
im.paste(pasted_image, (x,y)) # (x,y) coordinates of upper left corner as a tuple
im.save(filename)
im.show()
```

To complete Checkpoint 1, show a TA or a Mentor your code and the output of your program.

Checkpoint 2: Image correction

To start this part, you are going to create a new file called `check2_helper.py` and create a function in it called `make_square`. Your function takes as an argument an image object, crops it to make it square and returns the resulting image. Remember to import Image in your file.

In your function, you will check the size of the image and crop some part of it to make it into a square image. If the image is wider than longer (horizontal), then you must crop it along the x-axis. If the image is longer than wider (vertical), then you must crop it along the y-axis. Always start from the top left corner.

Check the course notes for finding the size of an image and cropping. Remember that the crop function returns a new image.

Now write some simple code to use this function to make one of the images from first checkpoint square and show. We recommend you try this with some of the images titled `inc*` as well to see if it works for horizontal images as well as vertical ones.

Here is how example test code should look:

```
im = Image.open('1.jpg')
imsquare = make_square(im)
imsquare.show()
```

You see that your function does not open an image, but takes an image as input argument, and returns a new image. Once you are convinced that your function works, leave nothing in the file but the function.

Copy your solution from checkpoint 1 to a new file called `check2.py`. Modify your code so that:

- You now import your own module `check2_helper.py`
- You use your own function from this module to make each image in the wallpaper square before resizing them.

To complete Checkpoint 2, show a TA or a Mentor your code and the output of your program. We will check that you are calling your module correctly, and that your program has correct structure.

Checkpoint 3 (option 1): Use an external source for images

Instead of using the files given to you, you can use external sources. To help with this, we have written a module called `panoramio.py` that finds images taken at a specific address (using images from Google maps). The module has a few functions that will be useful to you in this part:

```
import panoramio as pan
urls = pan.getPhotos('Eiffel Paris France',5)
# Return URLs for 5 pictures of Eiffel Tower as a tuple

im = pan.openphoto(urls[0]) # Return an image object for the first image URL
im.show()  ## view the image
```

The function `getPhotos()` will take an address and the number of desired images (5 in the above example) as arugments. It will find all photos near the given address (up to the given number), and return a list of URLs for these images. There may be no photos at a given address, especially if your address is not understood by the program. In this case, you will get an empty list.

(**Important Note:** We have not learned lists, but we are about to. For this problem you can treat a list *just like you would a tuple*. One thing we have not told you, though, is that you can get the length of a list or a tuple using the `len` function, just like you would a string.)

For each photo, the URL is a string like this:

`+u'http://mw2.google.com/mw-panoramio/photos/medium/59461095.jpg'+`

Don't worry about the `u` prefix, it just means the string is encoded in Unicode. It should not matter to you.

To open an image from a URL, we use the `openphoto` method from `panoramio.py`. It works similarly to `Image.open()` but for URLs rather than local files. Its use is defined in the box above. To see how this works, open up the file `test_panoramio.py`, look at it, and run it. Were you able to predict what was going to happen?

Your task now in this checkpoint is straightforward:

- Save your `check2` program out as `check3_1.py`.
- In `check3_1.py`, write code to ask the user for an address using `input` and find 4 images at this address using `panoramio`.
- If the returned list has at least 4 images, then construct the checkpoint 2 wallpaper using these images and show the result.
- Otherwise, tell the user that you could not find sufficient number of pictures.

To complete Checkpoint 3, show a TA or a Mentor your code and the output of your program.

Checkpoint 3 (option 2): A different layout for wallpaper

Here is a bit of challenge for you. We will change the wallpaper layout so that you put 6 images in a line (using images named `1.jpg,..,6.jpg`), alternating the vertical location as shown in the figure below. Each white box is an image.

To accomplish this, save your `check2.py` file into `check3_2.py`. Then:

- You will need to resize the images (no cropping) proportional to their original size so that their height is 256.
- Paste them in a wallpaper of size 1000, 360 (the first image starts at (31,20)).
- Images should have 10 pixels in between along the horizontal axis, and move up and down 40 pixels along the vertical axis.

If you do this, I promise you will get a cool looking wallpaper!

To complete Checkpoint 3, show a TA or a Mentor your code and the output of your program.