Objectives

After completing this unit and lab 09 you should be able to:
• Create programs that will open, modify and save images
• Use Python to modify and combine existing images with special effects
• Iterate over and manipulate two dimensional arrays
Images in Python

• The collections we have seen so far are one dimensional sequences
  – Identifying any element required only a single index
• Now we'll see two-dimensional data that look like tables and have rows and columns
  – These structures are often called two dimensional arrays
  – These structures are somewhat different than lists of lists
• In this unit we'll discuss a particular structure of such arrays that represent images
• Images are handled by an external Python package called Pillow
  – Pillow is a fork of an earlier (and apparently orphaned) package PIL (Python Image Library)
  – Pillow appears to be part of some versions of Anaconda (eg: Mac) but not others (eg: Windows) but it is easily installed
    • If it does not appear to be installed when you run your
Digital Images

- An Image consists of a collection of **pixels** (picture elements) arranged in rows and columns
- A picture is stored in a two-dimensional array of pixels
- Each pixel stores a color
- Color is represented in RGB (Red-Green-Blue) format which consist of a tuple of three numbers representing the amount of red, green and blue that is in the specific color
  - For instance:
    - black is (0, 0, 0)
    - white is (255, 255, 255)
    - pure red is (255, 0, 0), etc.
- Any color can be represented as a combination of red, green, and blue

Each color channel can be from 0 to 255

314 x 209 pixels
Digital Images (cont'd)

• Each pixel in an image is identified by its position which is defined by a tuple of two numbers (column, row)
• Here are the positions of the pixels in the following 5x5 image:

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>i</th>
<th>j</th>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

• To summarize:
  – a pixel in the $i$ column and $j$ row is identified by its positions $(i, j)$
  – the value of a pixel is a tuple $(r, g, b)$ where $r$, $g$, $b$ are integer values (from 0 to 255) representing the red, green and blue channels of the pixel's color
The Pillow Image Module

For our purposes we will only use the Image module of Pillow, which contains functions and a class to read, manipulate and write images.

- For more information, review the functions in these slides in the Pillow documentation:

Among the functions are:

• Create a new image:
  ```python
  img = Image.new("RGB", (width, height),color=0)
  ```
  - default color for every pixel is (0,0,0), i.e. black

• Open an image that is stored in a file myimage.jpg:
  ```python
  img = Image.open("myimage.jpg")
  ```
  - img now holds the image stored in the file and it can be processed

• Both of these functions return an object of type Image
The Pillow Image Class

When working with Pillow, each particular image will be an object instantiating the Image class (which is part of the Image module). This class (and hence its objects) has a number of methods and attributes which can be used to manipulate images. Examples include:

- Get the size of an image:
  
  \[(w, h) = \text{img.size}\]
  
  - then, \(w\) is the image width and \(h\) is its height

- Save an image to a file `yourimage.jpg`:
  
  \[\text{img.save("yourimage.jpg")}\]

- Display an image on the screen:
  
  \[\text{img.show()}\]

- Note: The method may not work well with IDEs (eg: Spyder) and certain OS's.
  
  - We recommend that you save the image to a file and then use a standard image viewer to examine the image
The Pillow Image Class (con't)

- Get a pixel from an image:
  ```python
color = image.getpixel((x,y))
```
  - then color is a color tuple

- Set the (x, y) pixel in an image:
  ```python
image.putpixel((x,y), color)
```

Alternatively:

- we can load the image into a two dimensional array using
  ```python
  pixels = image.load()
  ```

- and get or set the (x,y) pixel using
  ```python
  color = pixels[x, y]
pixels[x, y] = color
  ```

- See examples: cross1.py, cross2.py and maxval.py
More Image Processing Methods

The Image class has a variety of methods for manipulating images

- **img.copy()** => new image
  - creates a new copy of the image
- **img.crop((x1,y1, x2,y2))** => new image
  - returns a copy of the **box** defined by the left-upper (x1,y1) and right-lower (x2,y2) points
- **img.paste(another_image, box)**
  - pastes another image into this image at the region specified by the box
- **img.resize(size)** => image
  - returns a resized copy of the image
- **img.rotate(angle)** => image
  - returns a copy image rotated counter clockwise by the given degrees
- and a lot more .....
Looping Over Pixels

• We can iterate over all the pixels in an image by using the height and width as ranges.
  – Ex from cross1.py:

        image = Image.open(in_file)
        (width, height) = image.size

        for i in range(width) :
            image.putpixel((i, height//2), (0, 0, 0))

        for j in range(height) :
            image.putpixel((width//2, j), (0, 0, 0))
Examples

- `blendimage.py` blends two images at a given rate
- `embedat.py` embeds one image inside another

Try the examples we discussed in the class at home, using your own images
  - Modify the filename string(s) (e.g., `in_file`) in each Python script to be the name of your image (including the extension)

- Replacing pixels in an image
  - Replacement with `putpixel()`:
    - `cross1.py`
  - Replacement with `[]` indexing:
    - `cross2.py`

- Blending images: `blendimage.py`

- You can create whatever pixel combination you want
- More examples in Lab09.