COMP 101 – PROBLEM SOLVING WITH COMPUTING

PIXEL/IMAGE CHEAT SHEET

Important information on how to access pixels correctly in our "2D" pixel array with column and row specifications.

The reference for the Image module of the PIL can be found at:

http://www.pythonware.com/library/pil/handbook/image.htm

Note that API says the following for `im.size`:

> **size**
>
> **im.size** => (width, height)
>
> Image size, in pixels. The size is given as a 2-tuple (width, height).

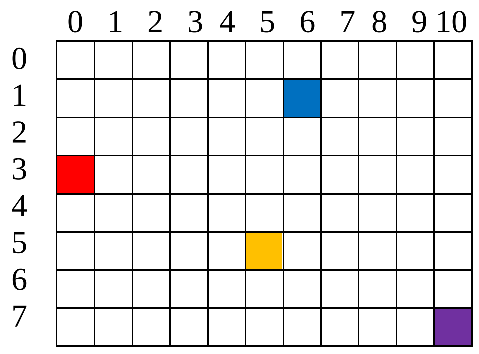
So `im.size[0]` => **columns** (width) and `im.size[1]` => **rows** (height). It also states:

> The access object behaves like a 2-dimensional array, so you can do:

```
pix = im.load()
print pix[x, y]
pix[x, y] = value
```

This means that the first value is the **column** and the second the **row**.

Our 'coordinate' system, where position `0, 0` (**column**, **row**) is in the top left corner. We have `11`(!) *columns* and `8`(!) *rows*. The red pixel is at position `(0, 3)`. The blue pixel at `(6, 1)`, the yellow one at `(5, 5)` and the bottom, right purple pixel is at position `(columns-1, rows-1)` – note that this notation is preferable to `(10, 7)` since it will work for any size image. (Careful, it's easy to be 'off-by-one')



Summary from slides:

- Pixel – picture element
- Pixel contains **8-bit** RGB values
- **8 bits** => 2**8 = 256 patterns, range 0 – 255
- **255 – max color value**, 0 – zero color value …
- RGB tuples (rvalue, gvalue, bvalue)

(255, 0, 0) - red	(0, 0, 127) - ?
(0, 0, 0) - ?	(0, 255, 255) - ?
(255, 255, 255) - ?	(127, 127, 127) - ?

Finally, note that if you have manipulated an image for a while, you may have to re-load it in order to start out with a ‘clean’ copy.

Basic Code Segment

```
import Image

im = Image.open(‘pic.jpg’)

print(‘image size: %d cols x %d rows’ % (im.size[0], im.size[1]))
print ‘mode: %s   format: %s’ %(im.mode, im.format)
im.show()

pix_ar = im.load()       # load image into 2D array
red_pixel = 255, 0, 0    # a red RGB pixel
cols = im.size[0]
rows = im.size[1]
```