**Pillow**

1. **import PIL**
2. PIL.\_\_version\_\_ # check the version using the version attribute.
3. help(PIL) # returns the object’s built-in documentation
4. **dir(**PIL**)**  # list the contents of an object (Classes)
5. **from PIL import Image** # the Image class is a **wrapper**
6. file="c://somFolder/ **someImage.gif** "

**image=Image.open(**file**)** # Image.open() is a function that loads an image from a file and returns an instance of Image Class

1. import inspect

**inspect.getmro( type( image ) )** # return a list of all of the classes that are being inherited

1. **image.show()** # Only If the image is **local** to the server
2. **from** **IPython.display** **import** **display** # renders remote image & display locally

**display(image**)

Explanation: Jupyter environment is running a special wrapper around the Python interpretor, called IPython. IPython allows the kernel back end to communicate with a browser front end, among other things. The IPython package has a display function which can take objects and use custom formatters in order to render them.

1. image.**copy()** # no arguments, return copy
2. image.**save(someImage.png)**  # save someImage**.gif as .png**

# The save method has a couple of parameters which are interesting. The first, called fp, is the **filename** we want to save the object too. The second, format, is interesting, it allows us to change the **type** of the image, but the docs tell us that this should be done **automatically** by looking at the file extension as well

1. **from PIL import** **ImageFilter** # To apply filters to the image
2. image = image**.convert( ‘RGB’ )** # before applying **filter** to the image needs to be

converted to the **RGB mode**

Explanation : Images like gifs are limited in how many colours can be displayed at once based on the size of the pallet. This is similar to a painter’s pallet, which only has so much room. This is actually a very old image file format. If we convert the image into something more sophisticated we can apply these interesting image transforms**.**

1. blurred\_image = image**.filter( PIL.ImageFilter.BLUR )** # use .filter() to apply filter(Here

**display( blurred\_image )** Blur)

1. Other Filters:
   1. CONTOUR
   2. DETAIL
   3. DGE\_ENHANCE
   4. EDGE\_ENHANCE\_MORE
   5. EMBOSS
   6. FIND\_EDGES
   7. SHARPEN
   8. SOOTH
   9. SMOOTH\_MORE
   10. BoxBlur( radius\_here )
       1. Blurs the image by setting each pixel to the average value of the pixels

in a square box extending radius pixels in each direction.

* 1. [check some demos here](https://hhsprings.bitbucket.io/docs/programming/examples/python/PIL/ImageFilter.html)

1. print(“ {} x {} “.format( **image.width , image.height** ) )
2. **image.crop( (50, 0, 190, 150) )** # returns copy of the image # no mutation

# Pass **Tuple** to image.crop() for cropping image

# The parameters are the coordinates of two points

# First point represent the **top-left corner** (50, 0)

# Second point represent the **bottom-right corner**. (190, 150)

1. To verify dimensions before applying, you can use ImageDraw. It outlines or draws over the image, highlighting the rectangular box (from the top left and bottom right coordinates)
2. **from** **PIL** **import** **ImageDraw**

drawing\_object=ImageDraw**.Draw(image)**

drawing\_object**.rectangle((50,0,190,150), fill = None, outline ='red')**

display(image)

1. One task that is fairly common in image and picture manipulation is to create **Contact Sheets** of images. A contact sheet is one image that actually contains several other different images. (So basically image are displayed over a contact sheet)
2. **from** **PIL** **import** **ImageEnhance**  # ImageEnhance has brightness object
3. enhancer=ImageEnhance**.Brightness( image )**

images=[]

for i in range(0, 10):

images.append( **enhancer.enhance( i/10 )** )

# we actually called the brightness routine by calling the enhance() function.

# It appends 10 copies of image with 10 brightness level

# We will display it as a palette by first defining Contact sheet to accommodate 10 images

1. **from PIL import Image**

first\_image = images[0]

contact\_sheet=PIL.Image**.new(**first\_image**.mode,** (first\_image.**width**,10\*first\_image.**height**)**)**

# The new function requires that we pass it a **mode**. We're going to use the mode **'RGB'** which stands for Red, Green, and Blue, and is the mode of our current first image. There are lots of different image mode formats, and this one is most common.

# For the size we have a **tuple**, which is the **width** of the image and the **height**. We'll use the width of our current first image, but for the **height we'll multiple this by ten**. This will make a sort of "canvas" for our contact sheet. Finally, the colour is optional, and we'll just leave it at black.

1. **current\_location = 0**

**for img in images**:

**# paste the current image into the contact sheet**

**contact\_sheet.paste(img, (0, current\_location) )** # starts from **top left corner**

**# And update the current\_location counter**

**current\_location=current\_location+450**

1. contact\_sheet = contact\_sheet**.resize((160,900) )** # **resize** the contact sheet for easy view

# resize take **int parameters**

**display(contact\_sheet)**

1. To display images in 3x3 matrix
2. **contact\_sheet=**

**PIL.Image.new( first\_image.mode, (first\_image.width\*3,first\_image.height\*3) )**

1. x=0

y=0

for img in images[1:]:

# Let’s paste the current image into the contact sheet

**contact\_sheet.paste( Img, (x, y) )**

# Now we update our X position. If it is going to be the width of the image, then we set it to 0 and update Y as well to point to the next "line" of the contact sheet.

if x+first\_image.width == contact\_sheet.width:

x=0

y=y+first\_image.height

else:

x=x+first\_image.width

1. **contact\_sheet = contact\_sheet.resize((int(contact\_sheet.width/2),int(contact\_sheet.height/2) ))**

**display(contact\_sheet)**

1. r, g, b = image.split() # returns RGB of the image
2. r = r.point( lambda x: x\*0.5)
3. image = Image.merge(‘RGB’, (r,g,b) )
4. text\_object = ImageDraw.Draw(image)
5. text\_object.text( (x , y) , text , font)