



Technical University of Lodz



Technical University of Łódź  
Institute of Electronics  
Medical Electronics Division

---

# IMAGE PROCESSING AND COMPUTER GRAPHICS

---

## Python Imaging Library 1

*Author:* MAREK KOCIŃSKI

**March 2010**

# 1 Purpose

To get acquainted with Python Imaging Library (PIL) and Python itself: loading images from file, saving images to file, basic operations and filtration.

## Time

4 × 45 minutes

# 2 Introduction

The Python Imaging Library (PIL) adds image processing capabilities to Python interpreter. This library supports many file formats, and provides powerful image processing and graphics capabilities. The most important class in the PIL is the Image class. This class is defined in the module in the same name.

# 3 Tasks

1. Open Python interpreter window (Start → Programy → EPD32-6.0.2 → IDLE)
2. It is convenient to create separate scripts for each processing task. Open new Editor Window (File → New Window) and write your code into it.
3. Import Image module: *import Image*
4. Load image *lenna.bmp*, display it and print basic information about it:

```
im = Image.open("lenna.bmp")
print im.format, im.size, im.mode
im.show()
```

5. Use *convert()* method to convert image representation from RGB to 8-bits gray-level *im\_L* (use "L" option as a method parameter). Find out about different image modes. Save result image into file

```
im_L.save("lenna-gray-level.png", "PNG")
```

6. Crop the region from the gray-level image and paste it into RGB image. Use *crop()*, *paste()* and *convert()* methods. Hint: define (left, upper, right, lower) coordinates of your region: *box=(100,100,400,400)*. It is possible to display region itself too. The results is presented in fig. 1.

```
box = (100,100,400,400)
region = im.crop(box)
region.show()
```

7. Blending between two images or regions is possible. Method *blend()* creates a new image by interpolating between two images or regions, using a constant alpha. Both images must have the same size and mode. Create second region box2=(200,200,500,500) and blend it with box. (Fig. 2)
8. Geometrical operations are available using functions: *transpose()*, *rotate()*. Read documentation of these functions, pay attention to possible input parameters. Transform loaded image into form presented in figure 3. Display **FULL** rotated image.
9. Load “flowers.bmp” image. Split it into three separate components R,G,B and change the band order to B,G,R (Fig. 4).

```
img = Image.open('flowers.bmp')
r,g,b = img.split()
img1 = Image.merge('RGB', (b,g,r))
img1.show()
```

10. Create new image (*Image.new()*) with the same size as *flowers.bmp*. Resize it (use one of the available interpolation methods) and divide into R,G,B bands. Copy each band into newly created image to achieve results presented in the figure 5. Check the difference among NEAREST, BILINEAR and BICUBIC interpolation methods.

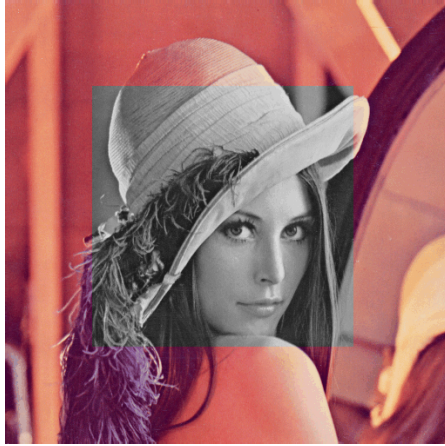


Figure 1: Crop—paste result.



Figure 2: Blending two regions with  $\alpha = 0.5$ .



Figure 3: Transformations: flipping and rotation of the image.



*RGB*



*BGR*

Figure 4: Image *Flowers* presented with different color components order



(a) Each band presented as gray-level image



(b) Each band presented as RGB image

Figure 5: R,G,B bands of *Flowers.bmp* presented as separate images