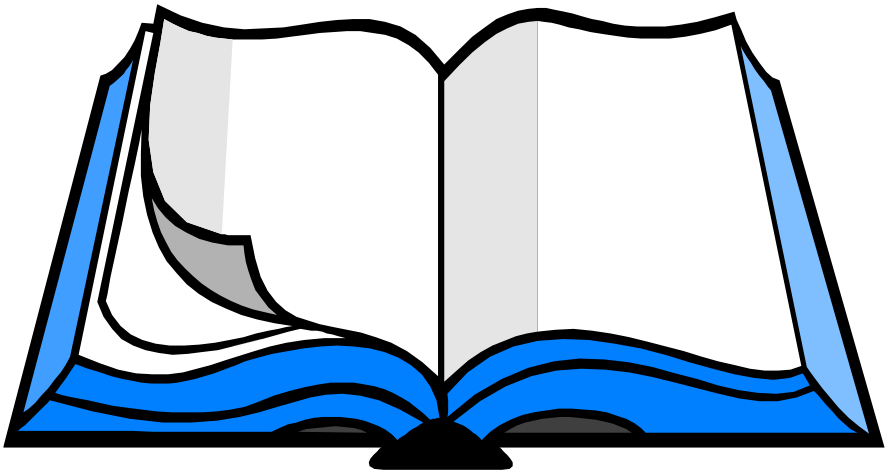
# University of Science

***Information Technology Department***

# PROJECT 02: COLOR PROCESSING




**Giảng viên**:

- Nguyễn Văn Quang Huy
- Ngô Đình Hy
- Nguyễn Đình Thúc

**Sinh viên:** 21127667 – Trương Công Gia Phát

# MỤC LỤC

# I. Mô tả project

Đồ án yêu cầu sử dụng thư viện PIL (open(), save() từ Image) để đọc và ghi, Matplotlib (imshow() từ pyplot) để hiển thị ảnh và sử dụng thư viện Numpy để lần lượt các chức năng xử lý ảnh sau:

1) Thay đổi độ sáng cho ảnh
2) Thay đổi độ tương phản
3) Lật ảnh (ngang – dọc)
4) Chuyển đổi RGB thành ảnh xám/sepia
5) Làm mờ/sắc nét ảnh
6) Cắt ảnh theo kích thước (cắt ở trung tâm)
7) Cắt ảnh theo khung hình tròn
8) Cắt ảnh theo khung là 2 hình elip chéo nhau (Không bắt buộc)

Các chức năng mà em đã hoàn thành:

- ✓ Thay đổi độ sáng cho ảnh
- ✓ Thay đổi độ tương phản
- ✓ Lật ảnh (ngang – dọc)
- ✓ Chuyển đổi RGB thành ảnh xám/sepia
- ✓ Làm mờ/sắc nét ảnh
- ✓ Cắt ảnh theo kích thước (cắt ở trung tâm)
- ✓ Cắt ảnh theo khung hình tròn
- ✓ Cắt ảnh theo khung là 2 hình elip chéo nhau (Không bắt buộc)

# II. Mô tả các hàm và ý tưởng thực hiện

## 1) Hàm đọc ảnh

**read_image(img_path)**

img_path là đường dẫn của ảnh cần đọc, hàm sẽ trả về một mảng numpy của ảnh đó.

## 2) Thay đổi độ sáng cho ảnh

### a) Ý tưởng thực hiện

Để tăng độ sáng cho ảnh, chúng ta cộng từng pixel của ma trận ảnh với một hệ số alpha nào đó. Ngược lại nếu muốn giảm độ sáng của ảnh ta sẽ trừ từng pixel với giá trị alpha.

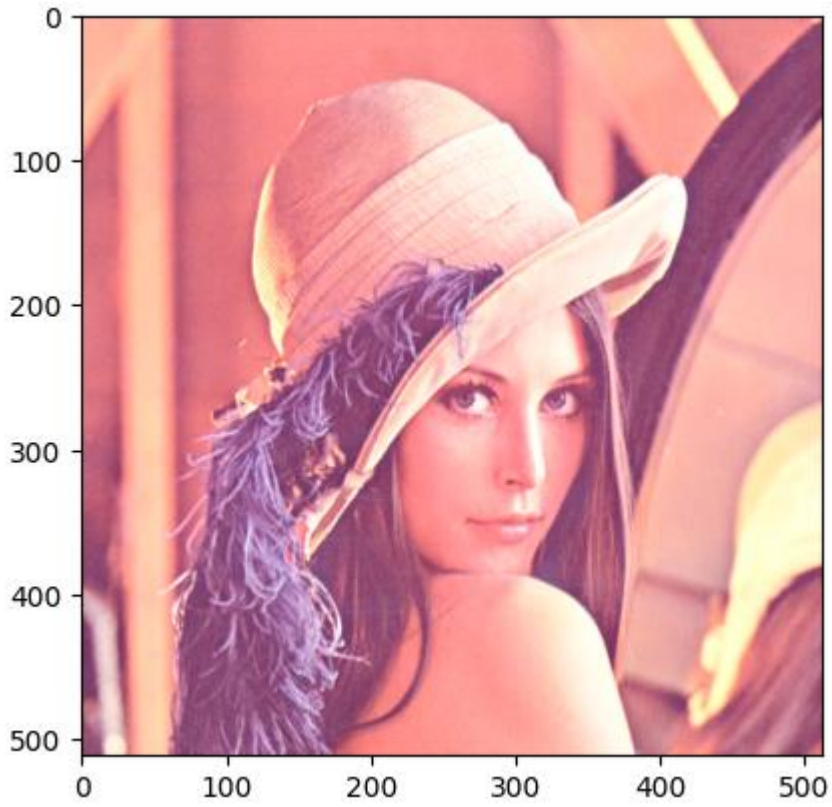
### b) Mô tả hàm

**brighten_image(img, alpha = 50)**

Hàm nhận img là ma trận ảnh và alpha là một con số bất kì. Ta tiến hành cộng từng pixel của ma trận với alpha bằng lệnh **img = img + float(alpha).** Sau đó giới hạn giá trị của các pixel trong khoảng [0; 255] bằng lệnh **np.clip(img, 0, 255).** Hàm sẽ trả về Image thuộc lớp PIL từ ma trận ảnh bằng lệnh **Image.fromarray(img.astype(np.uint8)).**

### c) Kết quả

Với alpha = 50 ta được:




## 3) Thay đổi độ tương phản

### a) Ý tưởng thực hiện

Đầu tiên tính hệ số tương phản bằng công thức:

$$F = \frac{259(C + 255)}{255(259 - C)}$$

Tiếp theo với hệ số tương phản F ta áp dụng công thức

$$R' = F(R - 128) + 128$$

Với R là các pixel trên ma trân ảnh.

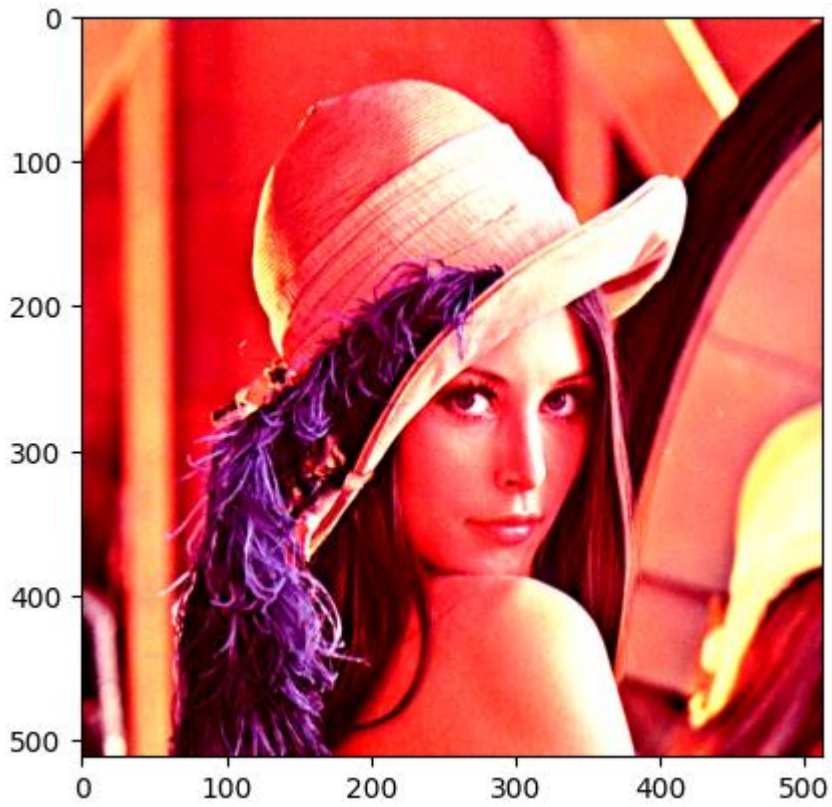
### b) Mô tả hàm

**adjust_contrast(img, alpha = 100)**

Hàm nhận vào ma trận ảnh img và hệ số alpha bất kì. Áp dụng công thức trên để tính hệ số tương phản với alpha bằng lệnh **factor = (259 * (255 + alpha)) / (255 * (259 - alpha)).** Sau đó tính toán trên từng pixel với lệnh **img = float(factor) * img - factor * 128 + 128**. Giới hạn giá trị của các pixel trong khoảng [0; 255] bằng **np.clip(img, 0, 255)**. Hàm trả về ảnh thuộc lớp PIL bằng **Image.fromarray(img.astype(np.uint8)).**

### c) Kết quả

Với alpha = 100 thu được kết quả:




## 4) Lật ảnh (ngang – dọc)

### a) Ý tưởng thực hiện

Để lật một ảnh theo hướng ngang hoặc dọc, ta chỉ cần lật ma trận ảnh tương ứng với ảnh đó theo trục (axis) 0 hoặc 1 trong Python.

### b) Mô tả hàm

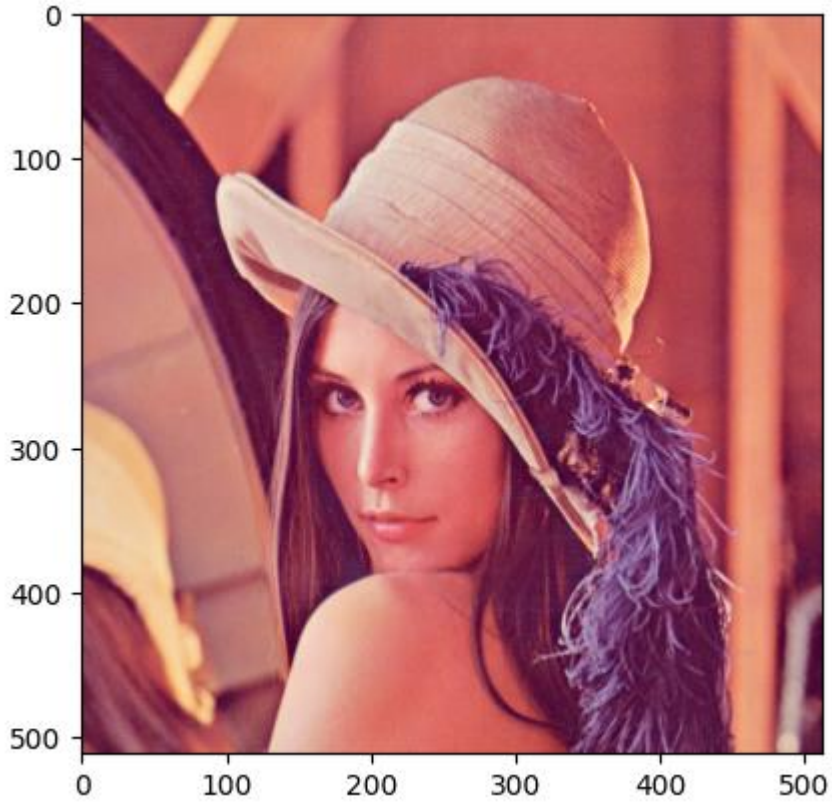
**def flip(img, axis)**

Hàm nhận vào ma trận ảnh img và trục (axis) mình cần lật. Đầu tiên ta đảm bảo giá trị của biến axis nằm trong khoảng số chiều của ảnh bằng lệnh **axis = axis % img.ndim**.

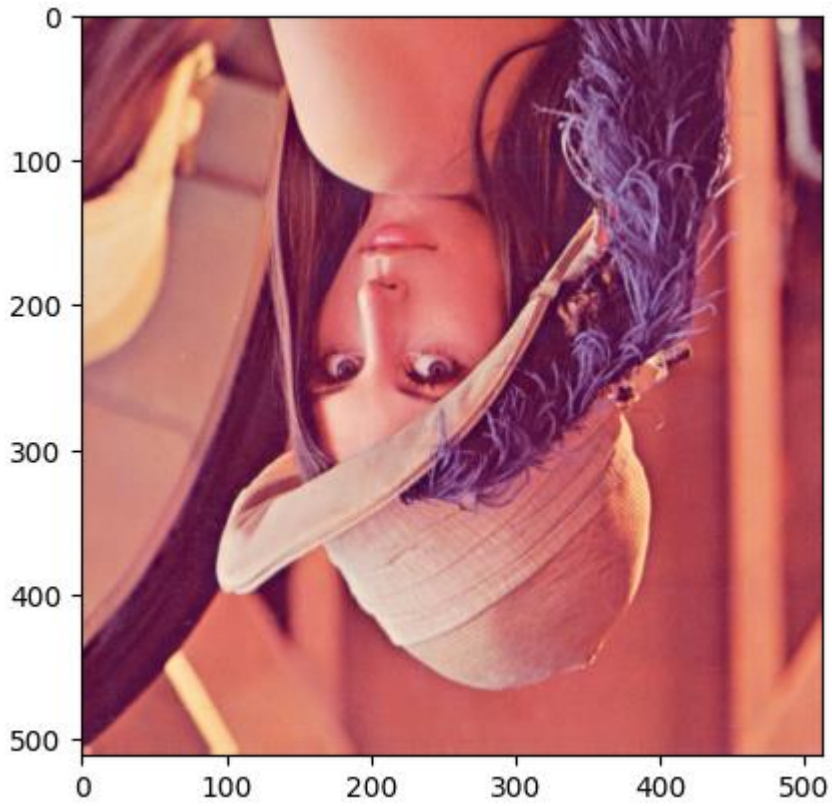
Tiếp theo tạo một danh sách arr với img.ndim phần tử, trong đó mỗi phần tử ban đầu là slice(None) bằng lệnh **new = [slice (None)] * img.ndim**. Lệnh **arr[axis] = slice(None, None, -1)** sẽ cập nhật arr bằng cách đặt slice trên trục chỉ định thành **slice(None, None, -1)** sẽ làm các phần tử trên trục được truy cập theo thứ tự ngược lại. Hàm trả về mảng đã được lật bằng lệnh **return img[tuple(arr)].**

### c) Kết quả

Lật ảnh theo chiều ngang (axis = 1)




Lật ảnh theo chiều dọc (axis = 0)




# 5) Chuyển đổi RGB thành ảnh xám/sepia

## a) Ý tưởng thực hiện

Để chuyển đổi sang ảnh xám ta nhân ma trận ảnh với ma trận **[0.3, 0.59, 0.11]** để tính trung bình các giá trị RGB tương ứng với mỗi pixel, từ đó giảm chiều màu xuống một chiều và ảnh sẽ thành ảnh xám. Còn để chuyển đổi sang ảnh sepia ta nhân ma trận ảnh với ma trận **[[0.393, 0.769, 0.189], [0.349, 0.686, 0.168], [0.272, 0.534, 0.131]].**

## b) Mô tả hàm

**def to_grayscale(img)**

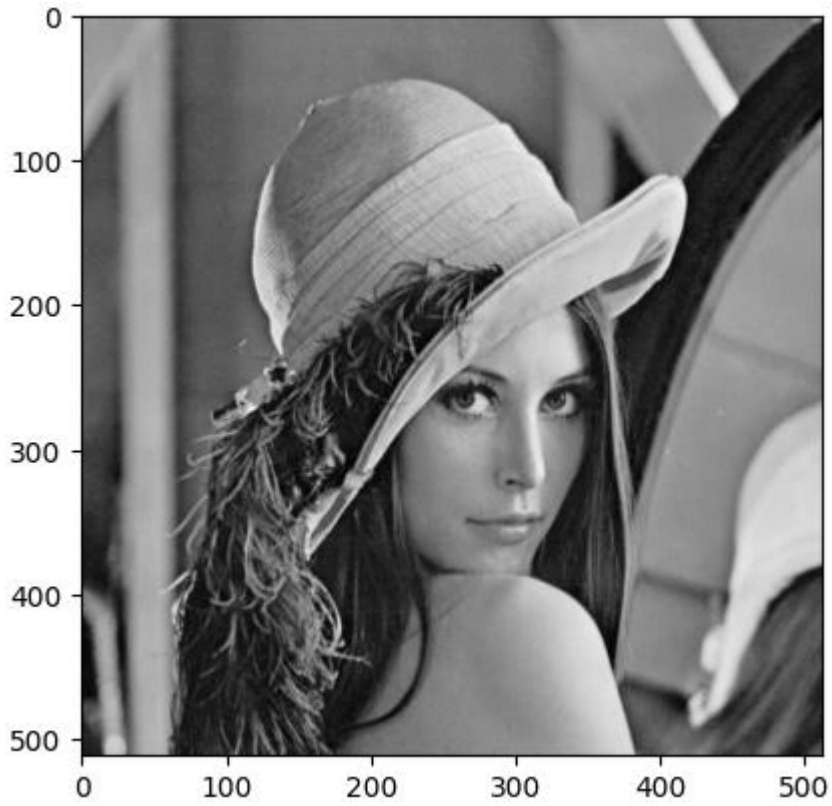
Hàm nhận vào ma trận ảnh img và nhân nó với ma trận gray_matrix bằng lệnh **img = img @ gray_matrix.** Hàm trả về ảnh thuộc lớp PIL bằng r**eturn Image.fromarray(img.astype(np.uint8)).**

**def to_sepia(img)**

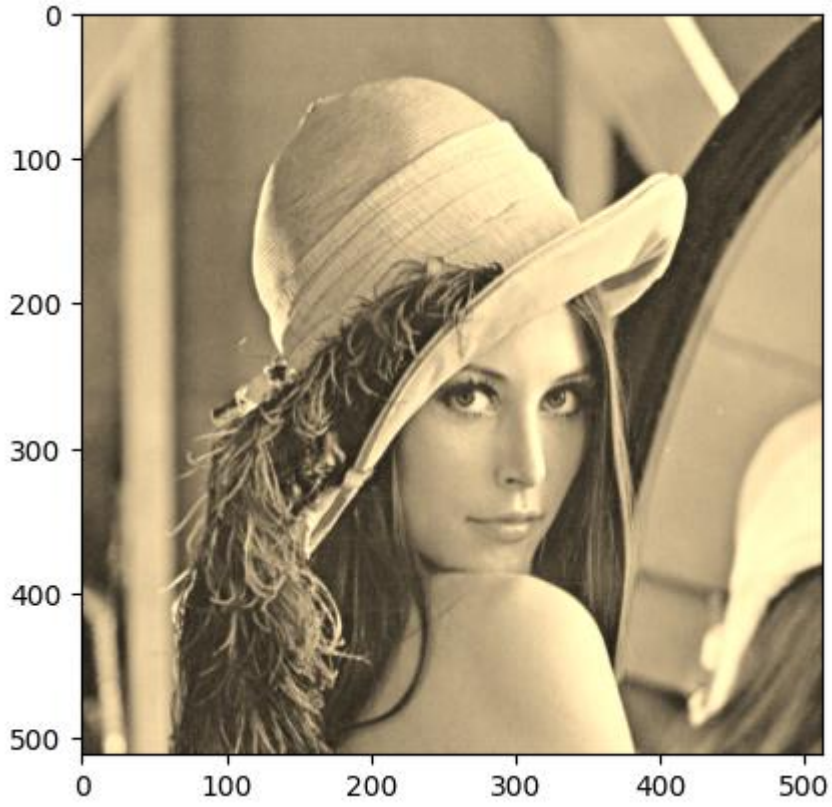
Hàm nhận vào ma trận img và nhân nó với ma trận sepia_matrix bằng lệnh **img = img @ sepia_matrix**. Sau đó giới hạn giá trị các pixel bằng lệnh **np.clip(img, 0, 255).** Hàm trả về ảnh thuộc lớp PIL bằng **Image.fromarray(img.astype(np.uint8)).**

## c) Kết quả

Ảnh xám:


0
100
200
300
400
500
0
100
200
300
400
500

Ảnh sepia:


0
100
200
300
400
500
0
100
200
300
400
500

# 6) Làm mờ/sắc nét ảnh

## a) Ý tưởng thực hiện

Ý tưởng chung của việc làm mờ ảnh và sắc nét ảnh là dùng phương pháp **convolution**. Ta sẽ dùng một ma trận kernel để thay đổi giá trị các pixel trong ảnh gốc bằng cách trượt kernel qua từng vùng nhỏ của ảnh gốc và tính toán lại giá trị của các pixel trong vùng đó. Đối với làm mờ, quá trình convolution sẽ tính toán lại giá trị trung bình của các pixel trong vùng, giá trị trung bình này sẽ thay thế giá trị pixel tại tâm vùng, gây hiệu ứng làm mờ. Còn đối với làm sắc nét, quá trình convolution sẽ tính toán lại giá trị cho pixel tại vị trí trung tâm của vùng, giá trị này thể hiện sự khác biệt giữa pixel tại vị trí trung tâm và các pixel xung quanh, từ đó làm sắc nét các cạnh và chi tiết trong ảnh.

## b) Mô tả hàm

### def blur_img(img)

Hàm nhận vào ma trận ảnh img, đầu tiên ta định nghĩa kernel là ma trận

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

**temp = np.zeros((img.shape[0] + 2, img.shape[1] + 2, img.shape[2]))** dùng để thêm một lớp viền có độ dày pixel vào ma trận ảnh img bằng cách tạo một mảng pad mới có kích thước lớn hơn img theo chiều dọc và ngang, nội dụng của ma trận img sẽ được sao chép vào giữa mảng pad. Tiếp theo ta thực hiện convolution bằng cách lặp qua từng pixel, đối với mỗi pixel ta sẽ lấy phần 3x3 tương ứng trong pad và nhân nó với kernel, cuối cùng gán lại kết quả vào vị trí của ma trận img. Hàm trả về ảnh của PIL bằng **return Image.fromarray(img.astype(np.uint8))**.

### def sharpen_img(img)

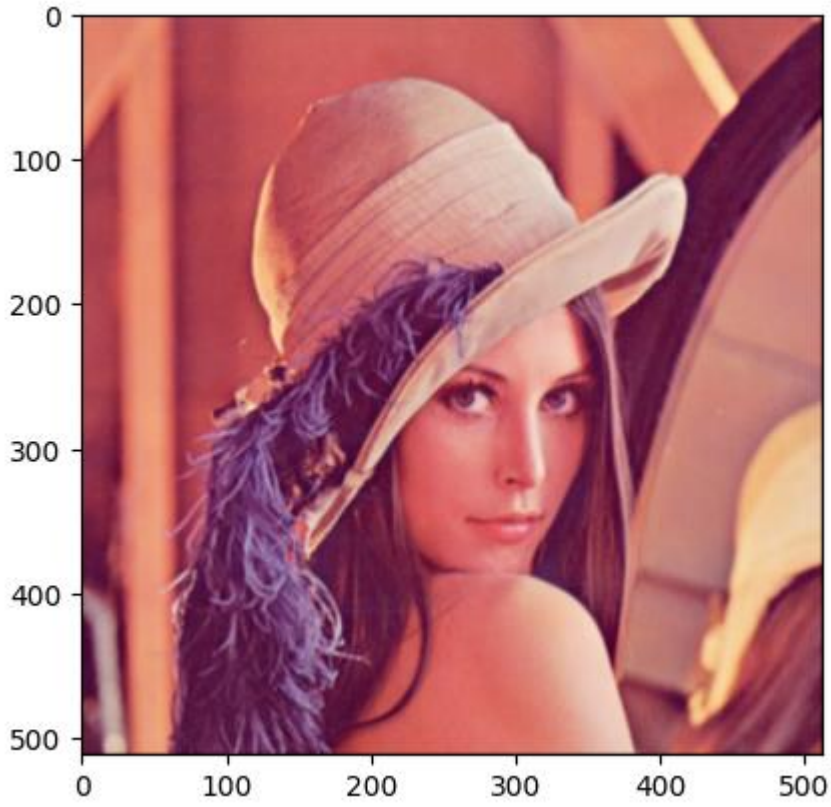
Hàm nhận vào ma trận ảnh img, kernel mà ta chọn là

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

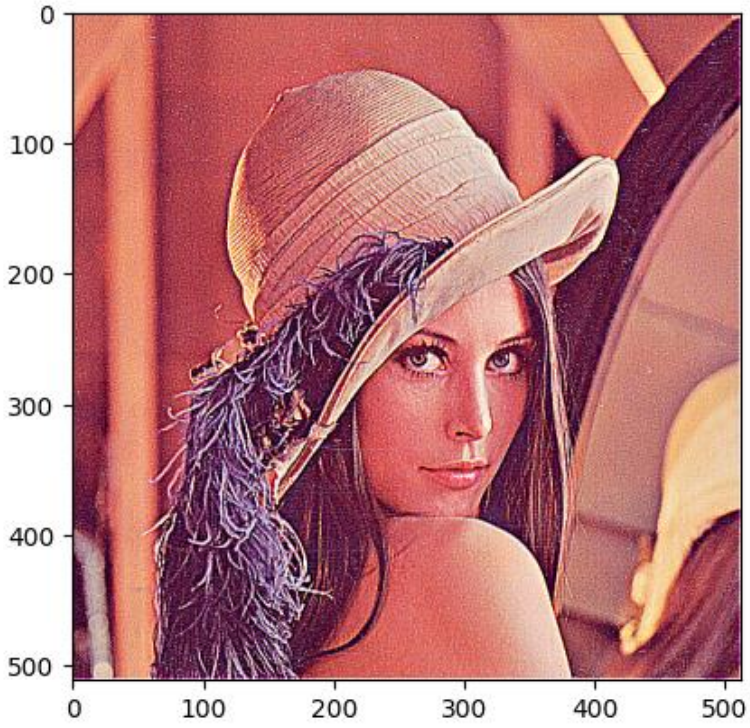
Ta cùng làm tương tự như bước làm mờ ảnh, chỉ khác ở việc trước khi xử lý ảnh ta sẽ chuẩn hoá nó bằng **img = img / 255.0**. Sau khi hoàn thành việc convolution sẽ nhân nó lại với 255 và dùng **np.clip(img, 0, 255)** để đảm ảnh nằm trong khoảng giá trị hợp lệ.

## c) Kết quả

Làm mờ ảnh




Làm sắc nét ảnh



## 7) Cắt ảnh theo kích thước (cắt ở trung tâm)

### a) Ý tưởng thực hiện

Để có thể cắt được ảnh tại vị trí trung tâm ta chỉ cần cắt ma trận ảnh đó ngay tại vị trí trung tâm.

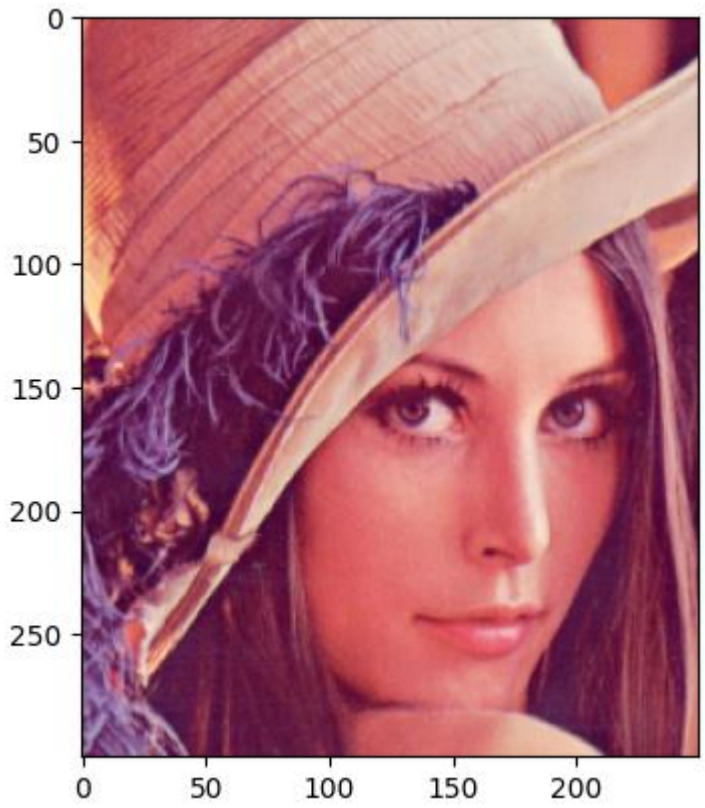
### b) Mô tả hàm

**def crop_center(image, x, y):**

Hàm nhận vào ma trận ảnh image và x, y đại diện cho kích thước cắt ảnh. Lần lượt lấy height và width của ma trận ảnh, sau đó tính toán vị trí cần cắt theo chiều ngang và chiều dọc bằng các lệnh **x_pos = width // 2 - (x // 2)** và **y_pos = height // 2 - (y // 2).** Hàm sẽ trả về ảnh đã được cắt bằng lệnh **return Image.fromarray(image[y_pos:y_pos + y, x_pos:x_pos + x])**

### c) Kết quả

Ảnh được cắt với x = 250, y = 300




## 8) Cắt ảnh theo khung hình tròn

### a) Ý tưởng thực hiện

Để cắt ảnh theo khung tròn, ta chỉ cần tạo một mặt nạ (mask) hình tròn và áp nó vào ảnh.

## b) Mô tả hàm

**def create_circularMask(img)**

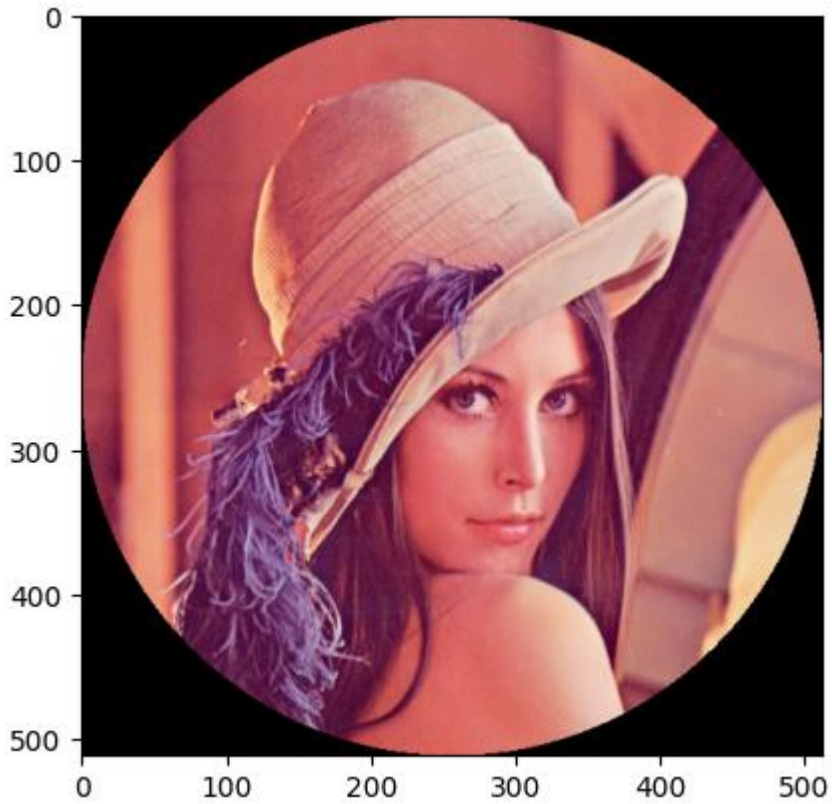
Hàm nhận vào ma trận ảnh img và lần lượt lấy ra chiều cao và chiều rộng của ảnh. Sau đó tính toán bán kính của đường tròn bằng cách chọn min của chiều rộng và chiều cao để đảm bảo đường tròn nằm trong ảnh bằng lệnh **radius = min(width, height)//2.**

Lấy ra tâm của đường tròn bằng lệnh **center = (height//2, width//2).** Sử dụng **np.ogrid()** để tạo ra 2 mảng y, x bằng câu lệnh **y, x = np.ogrid[:height, :width].** Mảng y chứa giá trị từ **0** đến **height – 1** và chứa các giá trị theo chiều dọc. Còn x chứa giá trị từ 0 đến width – 1 và chứa giá trị theo chiều ngang. Ta tạo ma trận **circleMask** có cùng kích thước với ảnh, chứa giá trị 0 ứng với điểm nằm ngoài đường tròn và 1 ứng với điểm nằm bên trong đường tròn bằng lệnh **circleMask = (x - center[1]) ** 2 + (y - center[0]) ** 2 <= radius ** 2** (phương trình đường tròn). Hàm sẽ trả về một mặt nạ hình tròn cho ảnh.

Để áp mặt nạ vào ảnh ta dùng câu lệnh:

**Image.fromarray(image*mask[:; :; np.newaxis])** với mask là mặt nạ hình tròn của ảnh ta gọi **mask[:; :; np.newaxis]** để thêm một chiều mới vào ma trận. Ma trận mới 3 chiều sẽ giúp cho ta áp mask vào ma trận ảnh**.** Sau đó ta nhân ma trận ảnh image vào mask để áp mặt nạ hình tròn lên ảnh.

## c) Kết quả



# 9) Cắt ảnh theo khung là 2 hình elip chéo nhau

## a) Ý tưởng thực hiện

Ta sẽ tạo ra 2 mặt nạ hình elip chéo nhau và cộng nó lại sau đó sẽ ra được mặt nạ hình bông hoa như yêu cầu. Áp nó vào ma trận ảnh để ra hình tương ứng.

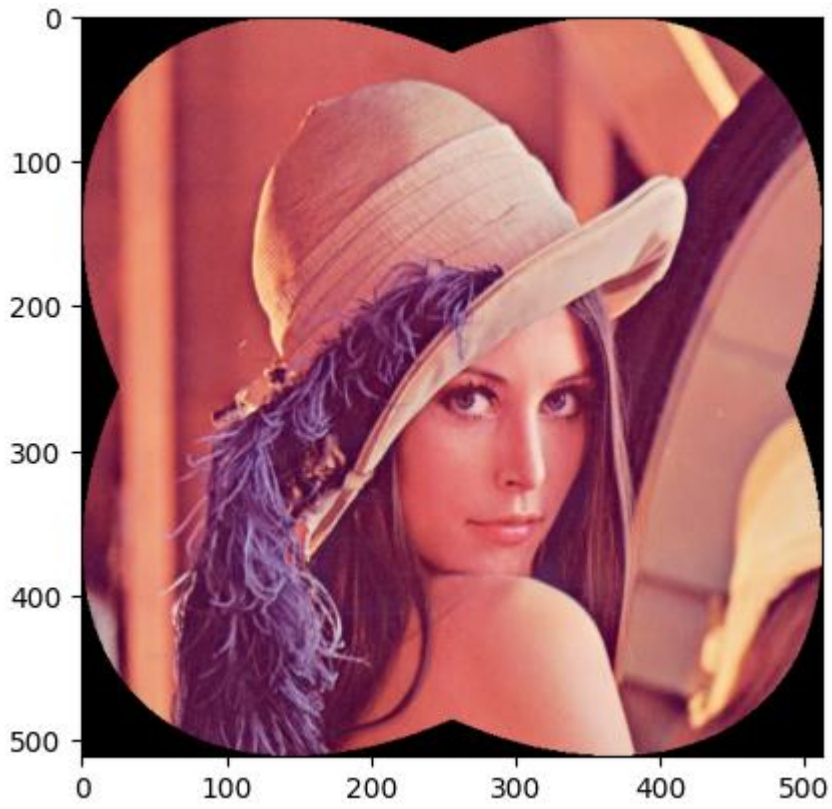
## b) Mô tả hàm

### def create_ellipseMask(img, angle):

Hàm nhận vào **img** là ma trận ảnh cần thêm khung và **angle** là góc ta cần xoay hình elip. Do hình tròn là dạng đặc biệt của elip nên những bước làm cũng tương tự như trên. Tuy nhiên do hình elip của chúng ta phải xoay theo góc angle nên ta phải tính toán lại x và y của elip bằng **new_x = (x - center[1]) * np.cos(angle) + (y - center[0]) * np.sin(angle)** và **new_y = (x - center[1]) * np.sin(angle) - (y - center[0]) * np.cos(angle)**. Áp dụng phương trình đường elip **ellipseMask = (new_x / a) ** 2 + (new_y / b) ** 2 <= 1** để tạo ra một mặt nạ có dạng elip. Hàm sẽ trả về mặt nạ có dạng elip.

Sau đó ta cũng tạo một mặt nạ elip khác ở góc **-angle** và cộng 2 mặt này lại với nhau thành mặt nạ 2 hình elip chéo nhau. Để áp mặt nạ này lên hình ta cũng sử dụng lệnh **Image.fromarray(image*mask[:; :; np.newaxis])**

## c) Kết quả



### 10) Hàm main

Hàm main sẽ cho người dung nhập đường dẫn ảnh và lựa chọn từ 1 đến 8 tương ứng với từng chức năng. Riêng các chức năng 3, 4, 5 sẽ cho người dùng chọn thêm cách xử lý ảnh đã cho.

## III. Nguồn tham khảo


https://github.com/kieuconghau/image-processing/blob/master/Source/18127259.ipynb?fbclid=IwAR1Y0kGMwODk6lZf-je3kVsjaKIBe4fX7IeV4M1-ygRJibbIpjsbQF40Ib0

https://quantrimang.com/hoc/ham-slice-trong-python-168764

https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/

https://towardsdatascience.com/image-processing-with-python-blurring-and-sharpening-for-beginners-3bcebec0583a

https://stackoverflow.com/questions/44865023/how-can-i-create-a-circular-mask-for-a-numpy-array