VIETNAM NATRIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

SUBJECT: Applied Mathematics and Statistics

CLASS 20CLC02

Phan Minh Phúc

20127063

Project 02: Image Processing

Lecturers: Vũ Quốc Hoàng, Nguyễn Văn Quang Huy, Lê Thanh Tùng, Phan Thị Phương Uyên

Ho Chi Minh City – 2022

CONTENTS

I. General information

- Project 02: Image Processing
- Author: Phan Minh Phúc – Student ID: 20127063
- Environment: jupyter notebook
- Programing language: python

II. Requirements

1. Project's requirement:

- Implement these image processing functions:
  - ➢ (1) Change image's brightness.
  - ➢ (2) Change image's contrast.
  - ➢ (3) Flip image vertically and horizontally.
  - ➢ (4) Convert RGB images to grayscale images.
  - ➢ (5) Combine 2 images of the same size into one. (grayscale images only)
  - ➢ (6) Blur image.
  - ➢ Provides a main () function that allows users to enter their image file's name and let them select which one of above image processing functions (from 1 to 6, 0 is for all of these functions) then return the result image with the name of the function respectively. (output format: png)
  - ➢ Optional:
    - Crop image circularly.
    - Crop image elliptically.

2. Report's requirement:

No.	Task	Status
1	Personal information	Done
2	Present your idea generally, describe all your defined functions	Done
3	Present the result of these function	Done
4	Your report must be paginated and included the reference section	Done

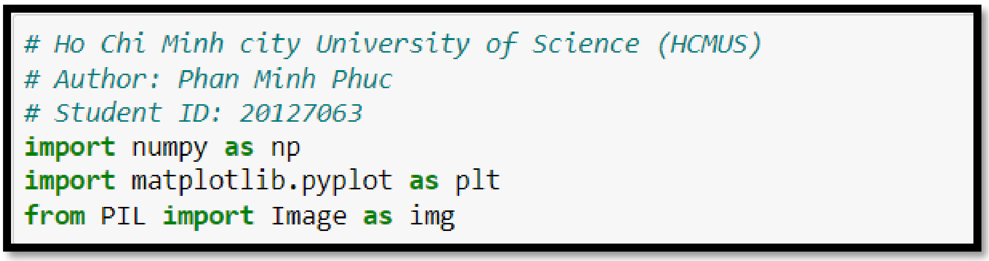
III. Conclusion

1. List of completed functions

No.	Completed functions	Status
1	Change image's brightness	Done
2	Change image's contrast	Done
3	Flip image vertically and horizontally	Done
4	Convert RGB images to gray images	Done
5	Combine 2 gray images of the same size into one	Done
6	Blur image	Done
7	Crop image circularly	Done
8	Crop image elliptically	Done

2. Present idea and describe functions

- Declare libraries:



```
# Ho Chi Minh city University of Science (HCMUS)
# Author: Phan Minh Phuc
# Student ID: 20127063
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image as img
```

- General idea:
  - ➢ Change image’s brightness: add a constant to every pixel of the image.
  - ➢ Change image’s contrast: multiply all current pixels by a constant.
  - ➢ Flip image vertically and horizontally: flip the current matrix of pixel vertically and horizontally.
  - ➢ Convert RGB images to grayscale images:
    - Average method: take the average of three colors. Since its an RGB image, so it means that you have add r with g with b and then divide it by 3 to get your desired grayscale image. Its done in this way: Grayscale = (R + G + B / 3). Since the three different colors have three different wavelength and have their own contribution in the formation of image, so we have to take average according to their contribution, not done it averagely using average method. Right now what we are doing is this, 33% of Red, 33% of Green, 33% of Blue. We are taking 33% of each, that means, each of the

portion has same contribution in the image. But in reality thats not the case. The solution to this has been given by luminosity (weighted) method.

- Weighted method: Since red color has more wavelength of all the three colors, and green is the color that has not only less wavelength then red color but also green is the color that gives more soothing effect to the eyes. It means that we have to decrease the contribution of red color, and increase the contribution of the green color, and put blue color contribution in between these two. So the new equation that form is: Grayscale = ((0.3 * R) + (0.59 * G) + (0.11 * B)). According to this equation, Red has contribute 30%, Green has contributed 59% which is greater in all three colors and Blue has contributed 11%.

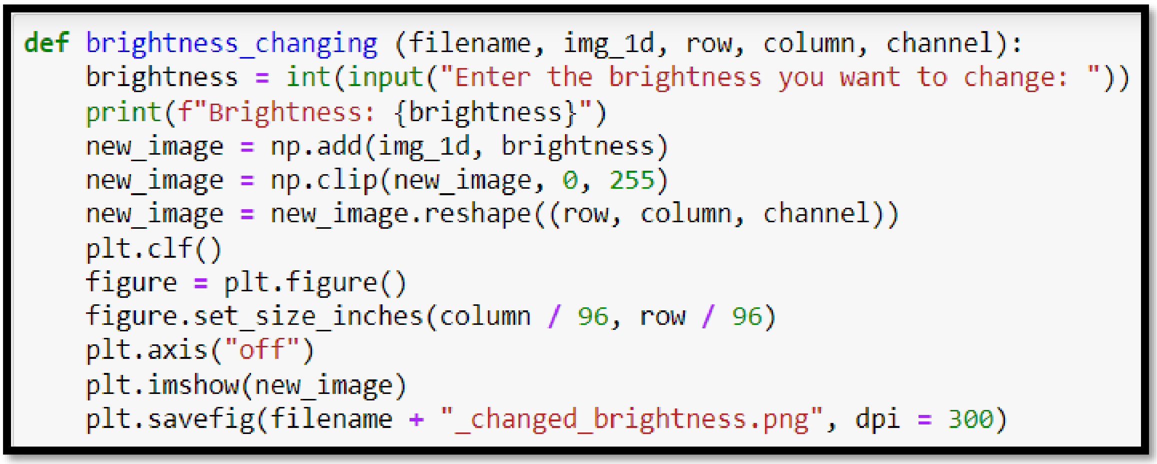
➢ Combine 2 images of the same size into one: combine the matrix of pixel of each image into one to get the final image.

➢ Blur image: apply the Gaussian filter to blur image. A Gaussian Filter is a low pass filter used for reducing noise (high frequency components) and blurring regions of an image. The filter is implemented as an Odd sized Symmetric Kernel (DIP version of a Matrix) which is passed through each pixel of the Region of Interest to get the desired effect. The kernel is not hard towards drastic color changed (edges) due to it the pixels towards the center of the kernel having more weightage towards the final value then the periphery. A Gaussian Filter could be considered as an approximation of the Gaussian Function (mathematics).

➢ Crop image circularly: create a circular mask, of which center is the center of the image, radius R, contains the index (in matrix) that belong to this circle. Every pixel which doesn’t belong to this mask is set to black.

➢ Crop image elliptically: create 2 elliptical masks, of which centers are the center of the image, semi-major axis a and semi-minor axis b, contains the index (in matrix) that belong to these ellipses, then rotate these masks pi / 4 and - pi / 4 respectively. Every pixel which doesn’t belong to this mask is set to black.

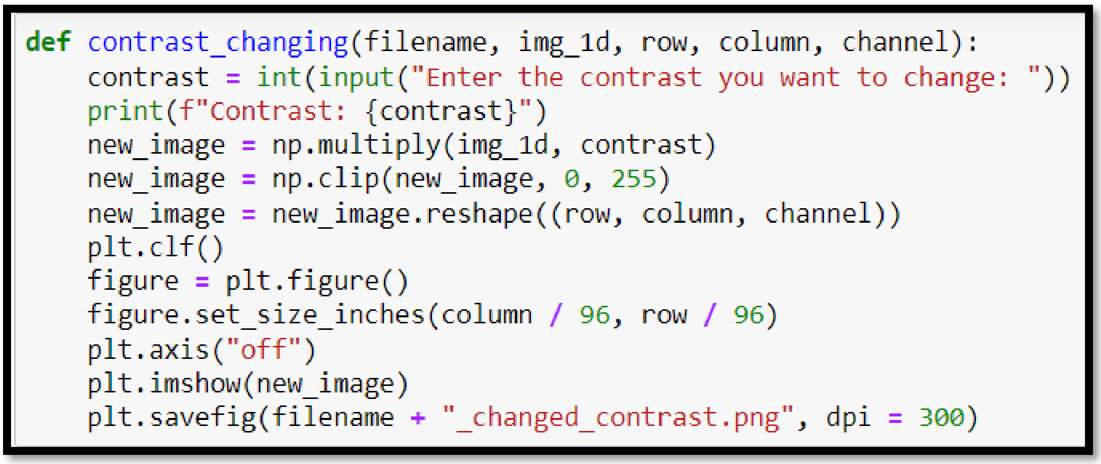
- Describe functions:
  - ➢ def brightness_changing (filename, img_1d, row, column, channel):



```
def brightness_changing (filename, img_1d, row, column, channel):
    brightness = int(input("Enter the brightness you want to change: "))
    print(f"Brightness: {brightness}")
    new_image = np.add(img_1d, brightness)
    new_image = np.clip(new_image, 0, 255)
    new_image = new_image.reshape((row, column, channel))
    plt.clf()
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(new_image)
    plt.savefig(filename + "_changed_brightness.png", dpi = 300)
```

- Allow users to enter the value of brightness they want to change.
- Use numpy.add() to add constant brightness to all pixels in img_id.
- Use numpy.clip() to limit the value of all pixels in the matrix between 0 and 255.
- Set the name of the output image as “{initial image’s name} + _changed_brightness.png”.

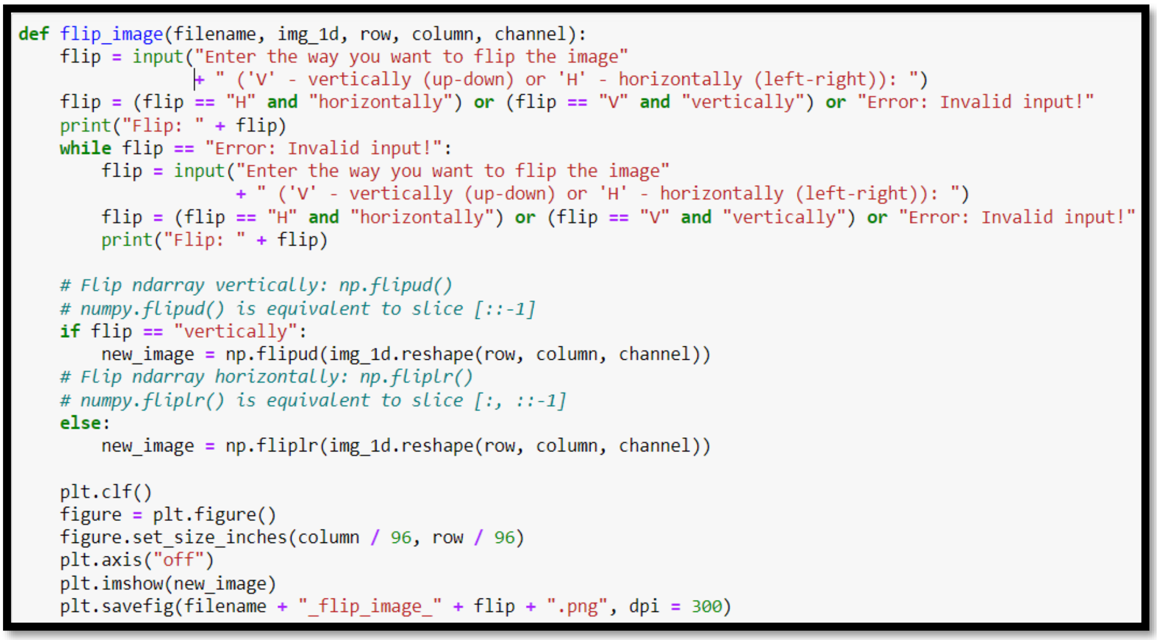
➢ def contrast_changing(filename, img_1d, row, column, channel):



```
def contrast_changing(filename, img_1d, row, column, channel):
    contrast = int(input("Enter the contrast you want to change: "))
    print(f"Contrast: {contrast}")
    new_image = np.multiply(img_1d, contrast)
    new_image = np.clip(new_image, 0, 255)
    new_image = new_image.reshape((row, column, channel))
    plt.clf()
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(new_image)
    plt.savefig(filename + "_changed_contrast.png", dpi = 300)
```

- Allow users to enter the value of contrast they want to change.
- Use numpy.multiply() to multiply all pixels in img_id by contrast.
- Use numpy.clip() to limit the value of all pixels in the matrix between 0 and 255.
- Set the name of the output image as “{initial image’s name} + _changed_ contrast.png”.

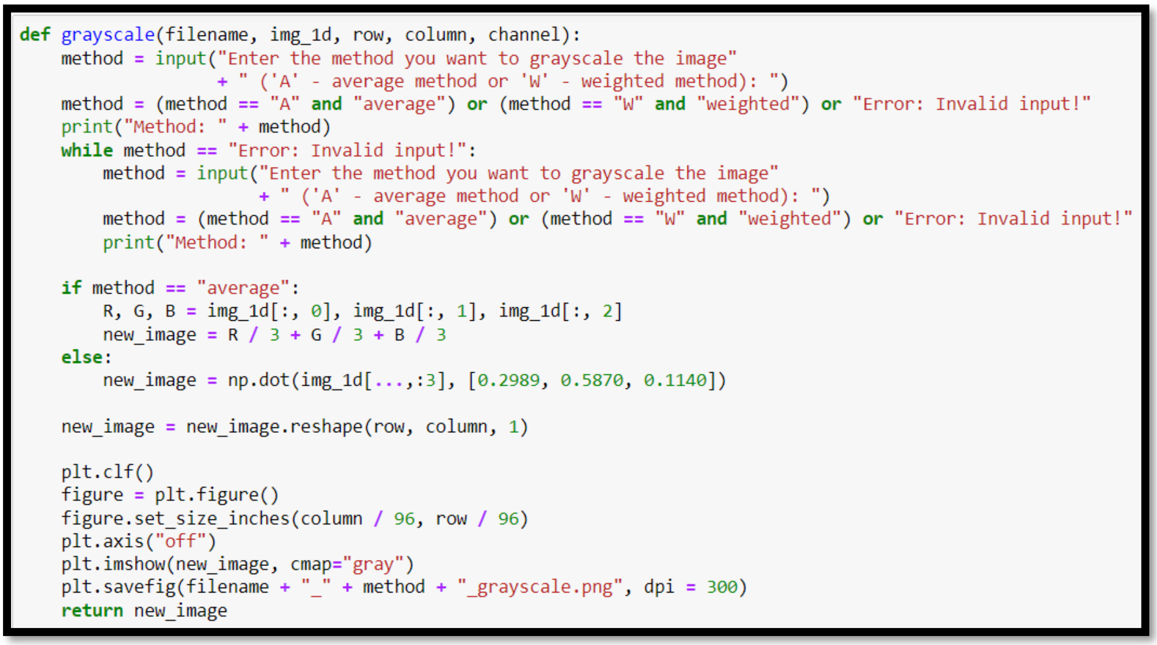
- def flip_image(filename, img_1d, row, column, channel):



```
def flip_image(filename, img_1d, row, column, channel):
    flip = input("Enter the way you want to flip the image"
                 + " ('V' - vertically (up-down) or 'H' - horizontally (left-right)): ")
    flip = (flip == "H" and "horizontally") or (flip == "V" and "vertically") or "Error: Invalid input!"
    print("Flip: " + flip)
    while flip == "Error: Invalid input!":
        flip = input("Enter the way you want to flip the image"
                     + " ('V' - vertically (up-down) or 'H' - horizontally (left-right)): ")
        flip = (flip == "H" and "horizontally") or (flip == "V" and "vertically") or "Error: Invalid input!"
        print("Flip: " + flip)

    # Flip ndarray vertically: np.flipud()
    # numpy.flipud() is equivalent to slice [::-1]
    if flip == "vertically":
        new_image = np.flipud(img_1d.reshape(row, column, channel))
    # Flip ndarray horizontally: np.fliplr()
    # numpy.fliplr() is equivalent to slice [:, ::-1]
    else:
        new_image = np.fliplr(img_1d.reshape(row, column, channel))

    plt.clf()
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(new_image)
    plt.savefig(filename + "_flip_image_" + flip + ".png", dpi = 300)
```

- Allow users to enter the way flip they want to flip the image (V or H).
- Process the input.
- If flip = “vertically”, use numpy.flipud() to flip image (matrix of pixel) vertically.
- If flip = “horizontally”, use numpy.fliplr() to flip image (matrix of pixel) horizontally.
- Set the name of the output image as “{initial image’s name} + _flip_image_ + flip + .png".

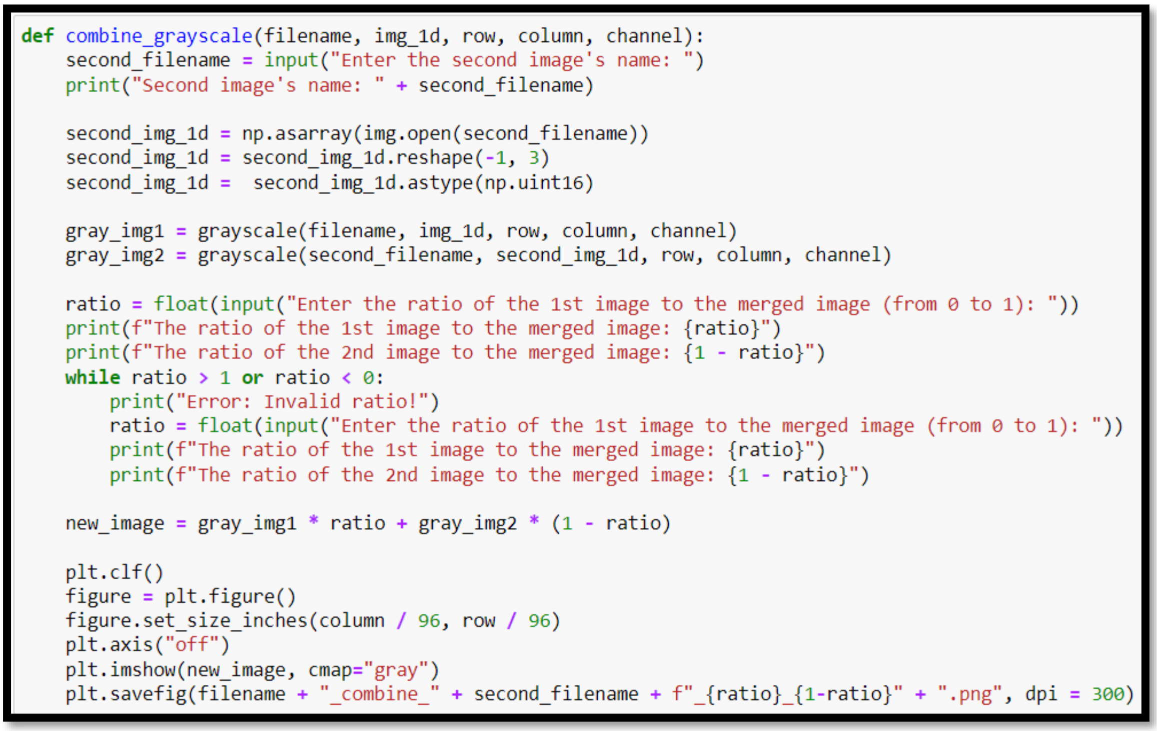
- def grayscale(filename, img_1d, row, column, channel):



```python
def grayscale(filename, img_1d, row, column, channel):
    method = input("Enter the method you want to grayscale the image"
                   + " ('A' - average method or 'W' - weighted method): ")
    method = (method == "A" and "average") or (method == "W" and "weighted") or "Error: Invalid input!"
    print("Method: " + method)
    while method == "Error: Invalid input!":
        method = input("Enter the method you want to grayscale the image"
                       + " ('A' - average method or 'W' - weighted method): ")
        method = (method == "A" and "average") or (method == "W" and "weighted") or "Error: Invalid input!"
        print("Method: " + method)

    if method == "average":
        R, G, B = img_1d[:, 0], img_1d[:, 1], img_1d[:, 2]
        new_image = R / 3 + G / 3 + B / 3
    else:
        new_image = np.dot(img_1d[...,:3], [0.2989, 0.5870, 0.1140])

    new_image = new_image.reshape(row, column, 1)

    plt.clf()
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(new_image, cmap="gray")
    plt.savefig(filename + "_" + method + "_grayscale.png", dpi = 300)
    return new_image
```

- Allow users to enter the method method of grayscaling the image (A or W).
- If method = “average”:
    - Assign R, G, B the first, the second and the third channel of all pixels.
    - Assign new_image the mean of sum of R, G and B.
- If method = “weighted”:
    - Assign new_image the result return after using numpy.dot() to multiply the first, the second and the third channel of all pixels by [0.2989, 0.5870, 0.1140] respectively.
- Set the name of the output image as “{initial image’s name} + _ + method + _grayscale.png".

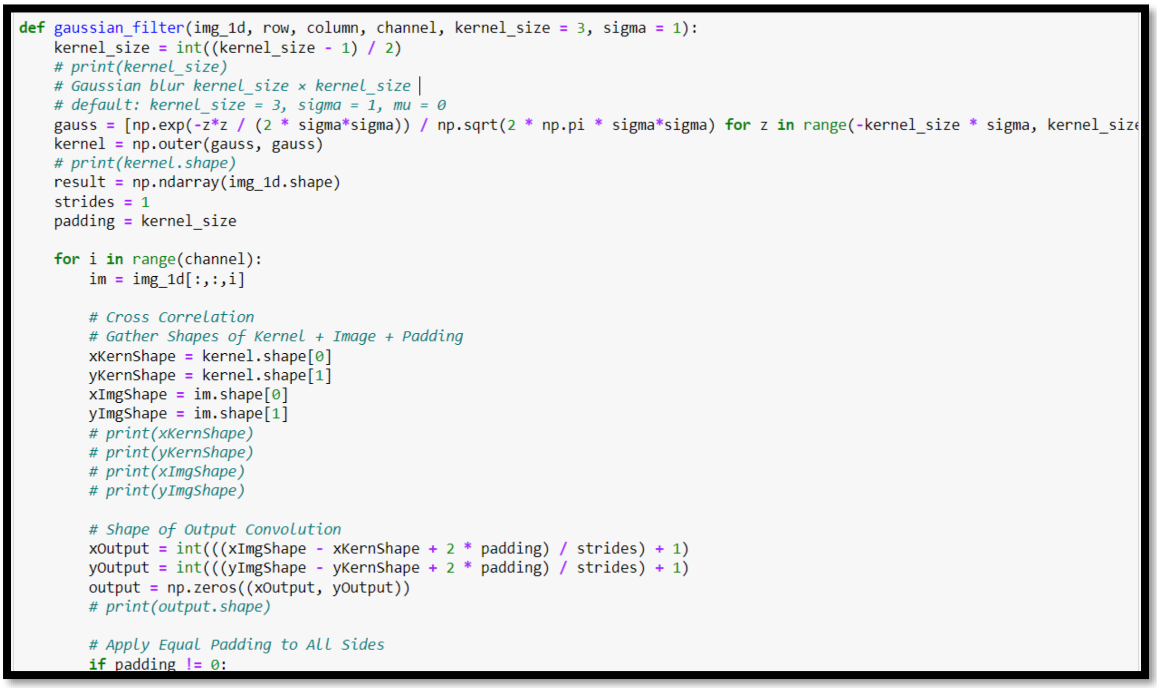
- def combine_grayscale(filename, img_1d, row, column, channel):



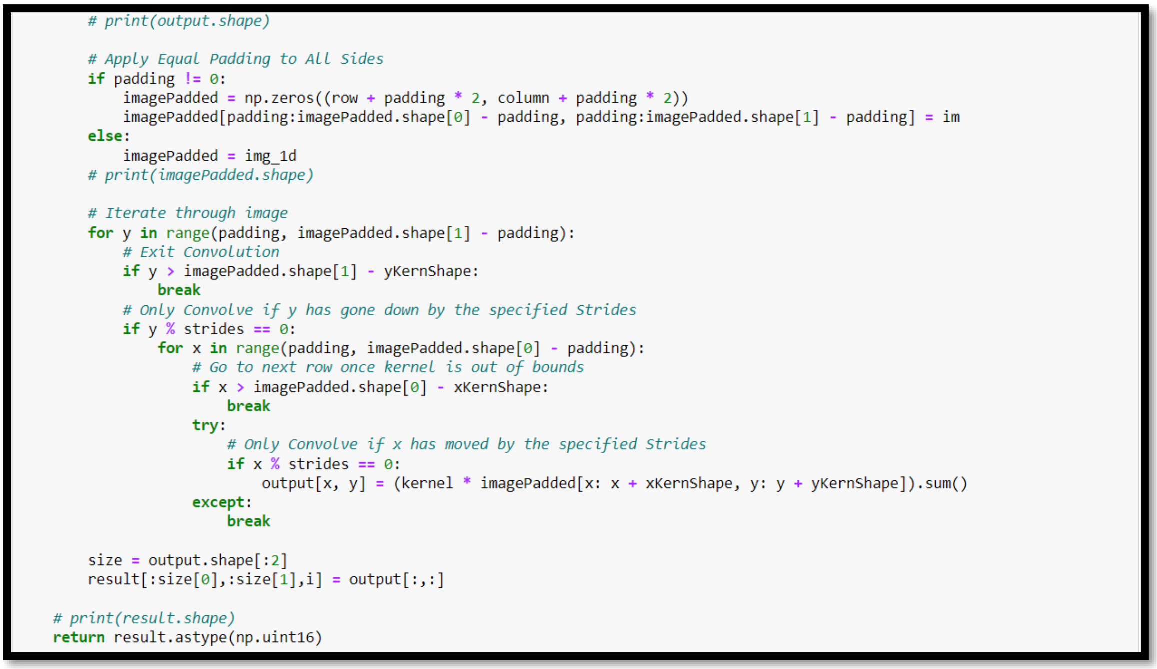
```
def combine_grayscale(filename, img_1d, row, column, channel):
    second_filename = input("Enter the second image's name: ")
    print("Second image's name: " + second_filename)

    second_img_1d = np.asarray(img.open(second_filename))
    second_img_1d = second_img_1d.reshape(-1, 3)
    second_img_1d =  second_img_1d.astype(np.uint16)

    gray_img1 = grayscale(filename, img_1d, row, column, channel)
    gray_img2 = grayscale(second_filename, second_img_1d, row, column, channel)

    ratio = float(input("Enter the ratio of the 1st image to the merged image (from 0 to 1): "))
    print(f"The ratio of the 1st image to the merged image: {ratio}")
    print(f"The ratio of the 2nd image to the merged image: {1 - ratio}")
    while ratio > 1 or ratio < 0:
        print("Error: Invalid ratio!")
        ratio = float(input("Enter the ratio of the 1st image to the merged image (from 0 to 1): "))
        print(f"The ratio of the 1st image to the merged image: {ratio}")
        print(f"The ratio of the 2nd image to the merged image: {1 - ratio}")

    new_image = gray_img1 * ratio + gray_img2 * (1 - ratio)

    plt.clf()
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(new_image, cmap="gray")
    plt.savefig(filename + "_combine_" + second_filename + f"_{ratio}_{1-ratio}" + ".png", dpi = 300)
```

- Allow users to enter the second image's name second_filename.
- Grayscale the initial image (the first image). Users are proactive in selecting the method of grayscaling the image. Assign the return result to gray_img1.
- Grayscale the second image. Users are proactive in selecting the method of grayscaling the image. Assign the return result to gray_img2.
- Allow users to enter ratio ratio of the 1st image to the merged image (from 0 to 1) and the remainder is the ratio of the second one.
- Assign new_image the result of adding gray_img1 and gray_img2 after multiplying them by their ratios.
- Set the name of the output image as "{the first image's name} + _combine_ {the second iamge's name} + _ + {ratio1} + _ {ratio2} + .png".

- def gaussian_filter(img_1d, row, column, channel, kernel_size = 3, sigma = 1):



```
def gaussian_filter(img_1d, row, column, channel, kernel_size = 3, sigma = 1):
    kernel_size = int((kernel_size - 1) / 2)
    # print(kernel_size)
    # Gaussian blur kernel_size x kernel_size
    # default: kernel_size = 3, sigma = 1, mu = 0
    gauss = [np.exp(-z*z / (2 * sigma*sigma)) / np.sqrt(2 * np.pi * sigma*sigma) for z in range(-kernel_size * sigma, kernel_size
    kernel = np.outer(gauss, gauss)
    # print(kernel.shape)
    result = np.ndarray(img_1d.shape)
    strides = 1
    padding = kernel_size

    for i in range(channel):
        im = img_1d[:,:,i]

        # Cross Correlation
        # Gather Shapes of Kernel + Image + Padding
        xKernShape = kernel.shape[0]
        yKernShape = kernel.shape[1]
        xImgShape = im.shape[0]
        yImgShape = im.shape[1]
        # print(xKernShape)
        # print(yKernShape)
        # print(xImgShape)
        # print(yImgShape)

        # Shape of Output Convolution
        xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
        yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
        output = np.zeros((xOutput, yOutput))
        # print(output.shape)

        # Apply Equal Padding to All Sides
        if padding != 0:
```



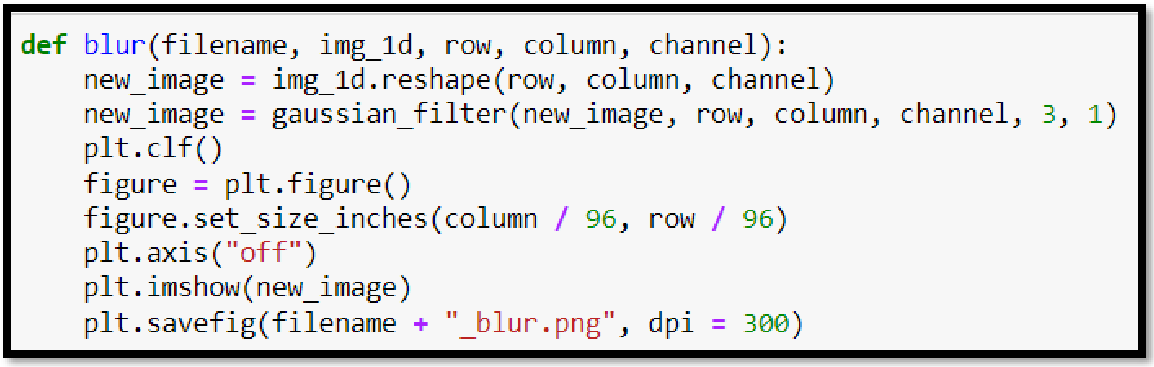
```
        # print(output.shape)

        # Apply Equal Padding to All Sides
        if padding != 0:
            imagePadded = np.zeros((row + padding * 2, column + padding * 2))
            imagePadded[padding:imagePadded.shape[0] - padding, padding:imagePadded.shape[1] - padding] = im
        else:
            imagePadded = img_1d
        # print(imagePadded.shape)

        # Iterate through image
        for y in range(padding, imagePadded.shape[1] - padding):
            # Exit Convolution
            if y > imagePadded.shape[1] - yKernShape:
                break
            # Only Convolve if y has gone down by the specified Strides
            if y % strides == 0:
                for x in range(padding, imagePadded.shape[0] - padding):
                    # Go to next row once kernel is out of bounds
                    if x > imagePadded.shape[0] - xKernShape:
                        break
                    try:
                        # Only Convolve if x has moved by the specified Strides
                        if x % strides == 0:
                            output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + yKernShape]).sum()
                    except:
                        break

        size = output.shape[:2]
        result[:size[0],:size[1],i] = output[:,:]

    # print(result.shape)
    return result.astype(np.uint16)
```

- Redefine the value of kernel_size.
- Calculate the value of Gaussian filter gauss for kernel_size based on this equation:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/(2\sigma^2)}$$

- Generate a matrix kernel size kernel_size x kernel_size based on the result of gauss by using numpy.outer().
- For each pixel's chanel:
  - Gather Shapes of Kernel, Image and Padding.
  - Generate the shape of Output Convolution.
  - If padding = 0, assign imagePadded the value of img_1d.
  - Else, assign imagePadded the value of img_1d padded by padding black row, column.
  - Start the convolution progress.
- Return the result after the convolution progress's been done.

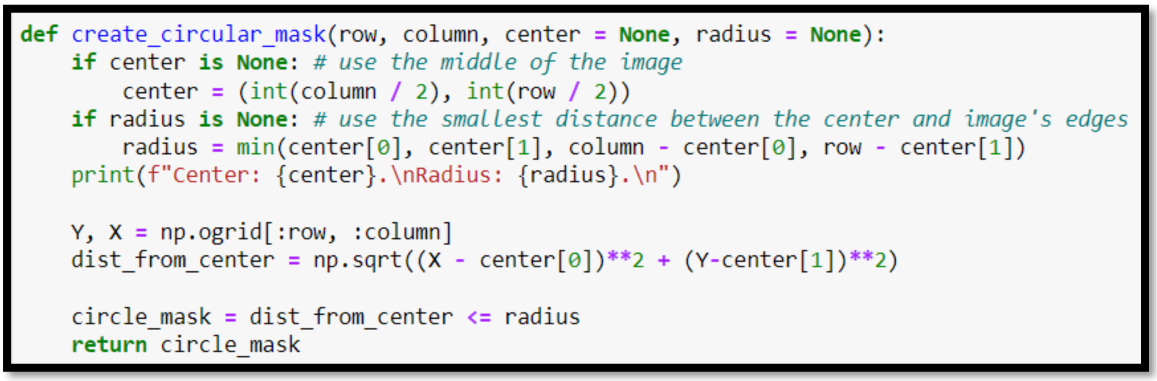
➢ def blur(filename, img_1d, row, column, channel):



```
def blur(filename, img_1d, row, column, channel):
    new_image = img_1d.reshape(row, column, channel)
    new_image = gaussian_filter(new_image, row, column, channel, 3, 1)
    plt.clf()
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(new_image)
    plt.savefig(filename + "_blur.png", dpi = 300)
```

- Assign new_image the result of gaussian_filter(img_1d, row, column, channel, 3, 1).
- Set the name of the output image as "{initial image's name} + _blur.png".

➢ def create_circular_mask(row, column, center = None, radius = None):

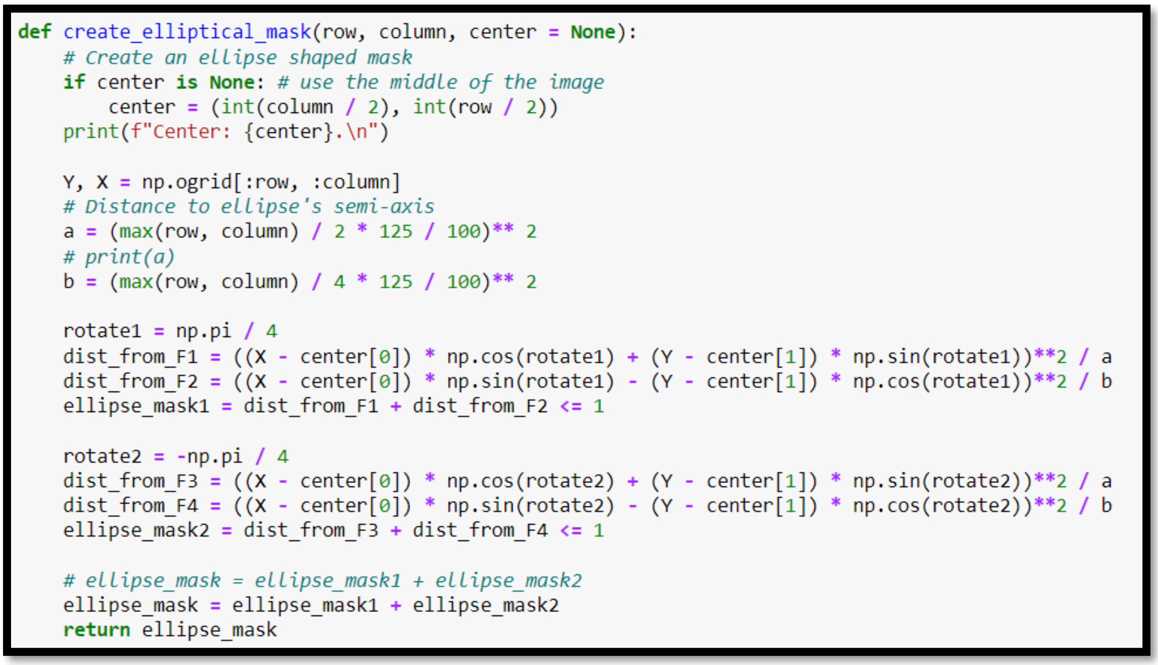


```
def create_circular_mask(row, column, center = None, radius = None):
    if center is None: # use the middle of the image
        center = (int(column / 2), int(row / 2))
    if radius is None: # use the smallest distance between the center and image's edges
        radius = min(center[0], center[1], column - center[0], row - center[1])
    print(f"Center: {center}.\nRadius: {radius}.\n")

    Y, X = np.ogrid[:row, :column]
    dist_from_center = np.sqrt((X - center[0])**2 + (Y-center[1])**2)

    circle_mask = dist_from_center <= radius
    return circle_mask
```

- Assign center the center of the image if center's input is None.
- Assign radius the smallest distance between center and image's edges if radius's input is None.
- Assign Y, X all index in image's row and column respectively by using numpy.ogrid().

- Assign disk_from_center the distance between each pixel and center.
- Assign circle_mask pixels belong to circle, of which center is center, whose distance <= radius radius. Return circle_mask.

➢ def create_elliptical_mask(row, column, center = None):



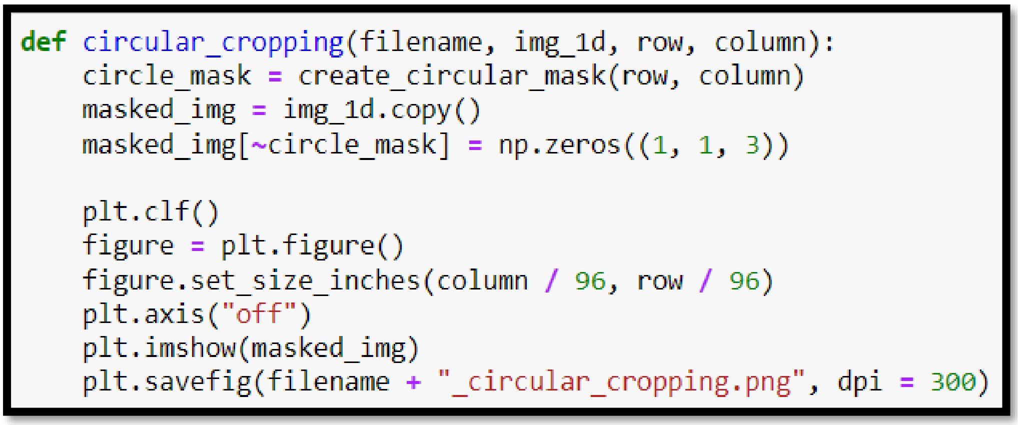
```
def create_elliptical_mask(row, column, center = None):
    # Create an ellipse shaped mask
    if center is None: # use the middle of the image
        center = (int(column / 2), int(row / 2))
    print(f"Center: {center}.\n")

    Y, X = np.ogrid[:row, :column]
    # Distance to ellipse's semi-axis
    a = (max(row, column) / 2 * 125 / 100)** 2
    # print(a)
    b = (max(row, column) / 4 * 125 / 100)** 2

    rotate1 = np.pi / 4
    dist_from_F1 = ((X - center[0]) * np.cos(rotate1) + (Y - center[1]) * np.sin(rotate1))**2 / a
    dist_from_F2 = ((X - center[0]) * np.sin(rotate1) - (Y - center[1]) * np.cos(rotate1))**2 / b
    ellipse_mask1 = dist_from_F1 + dist_from_F2 <= 1

    rotate2 = -np.pi / 4
    dist_from_F3 = ((X - center[0]) * np.cos(rotate2) + (Y - center[1]) * np.sin(rotate2))**2 / a
    dist_from_F4 = ((X - center[0]) * np.sin(rotate2) - (Y - center[1]) * np.cos(rotate2))**2 / b
    ellipse_mask2 = dist_from_F3 + dist_from_F4 <= 1

    # ellipse_mask = ellipse_mask1 + ellipse_mask2
    ellipse_mask = ellipse_mask1 + ellipse_mask2
    return ellipse_mask
```

- Assign center the center of the image if center's input is None.
- Assign Y, X all index in image's row and column respectively by using numpy.ogrid().\
- Assign a the square of 125% of max between row and column divided by 2.
- Assign b the square of 125% of min between row and column divided by 4.
- Generate 2 ellipse and flip them numpy.py / 4, - numpy.pi / 4 respectively based on this equation:

$$\frac{((x-h)\cos(A)+(y-k)\sin(A))^{2}}{a^{2}}+\frac{((x-h)\sin(A)-(y-k)\cos(A))^{2}}{b^{2}}=1$$

  - where h, k and a, b are the shifts and semi-axis in the x and y directions respectively and A is the angle measured from x axis.
  - Assign ellipse_mask1 and ellipse_mask2 pixels belong to ellipse, of which center is center, whose distance <= 1.
- Assign ellipse_mask the total of ellipse_mask1 and ellipse_mask2.

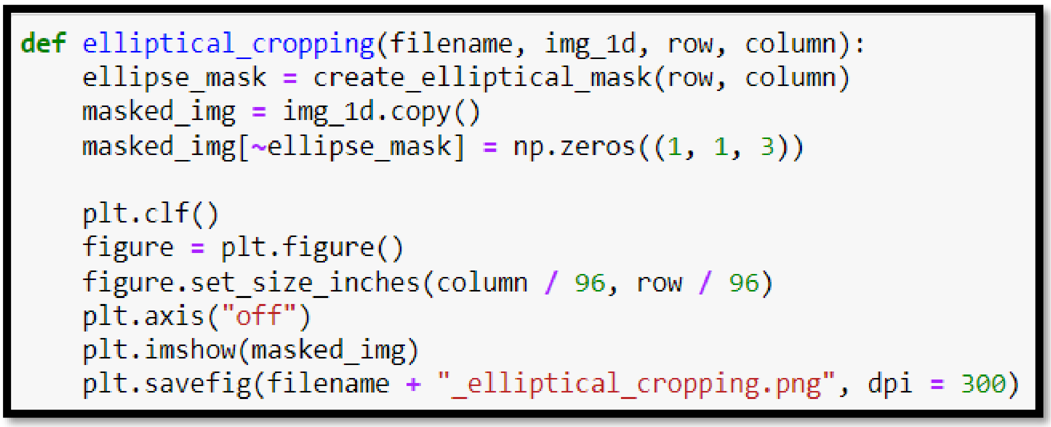
- def circular_cropping(filename, img_1d, row, column):



```
def circular_cropping(filename, img_1d, row, column):
    circle_mask = create_circular_mask(row, column)
    masked_img = img_1d.copy()
    masked_img[~circle_mask] = np.zeros((1, 1, 3))

    plt.clf()
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(masked_img)
    plt.savefig(filename + "_circular_cropping.png", dpi = 300)
```

  - Assign circle_mask the result of create_circular_mask(row, column).
  - Use numpy.copy() to copy img_1d to masked_img.
  - Use "~" to negate the value of
  - Set all pixels in masked_img, of which index equal to index of pixels in circle_mask whose value is true, to black.

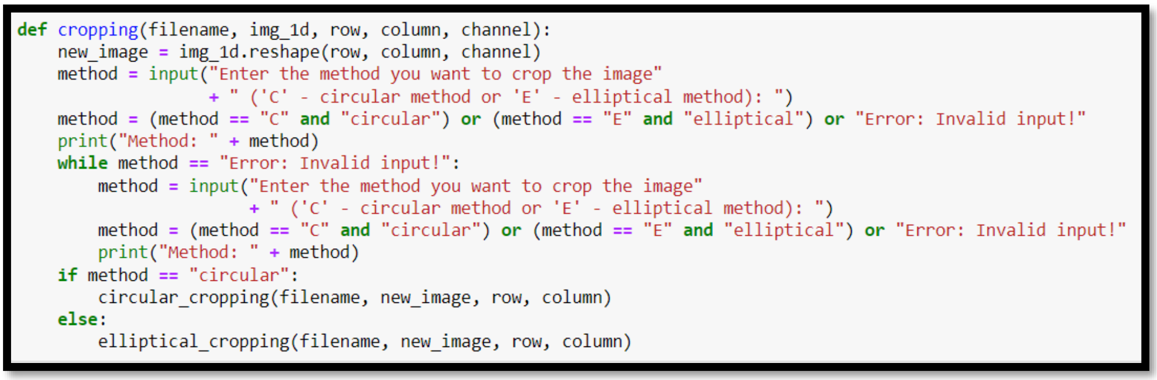
- def elliptical_cropping(filename, img_1d, row, column):



```
def elliptical_cropping(filename, img_1d, row, column):
    ellipse_mask = create_elliptical_mask(row, column)
    masked_img = img_1d.copy()
    masked_img[~ellipse_mask] = np.zeros((1, 1, 3))

    plt.clf()
    figure = plt.figure()
    figure.set_size_inches(column / 96, row / 96)
    plt.axis("off")
    plt.imshow(masked_img)
    plt.savefig(filename + "_elliptical_cropping.png", dpi = 300)
```

  - Assign ellipse_mask the result of create_elliptical_mask(row, column).
  - Use numpy.copy() to copy img_1d to masked_img.
  - Use "~" to negate the value of
  - Set all pixels in masked_img, of which index equal to index of pixels in ellipse_mask whose value is true, to black.

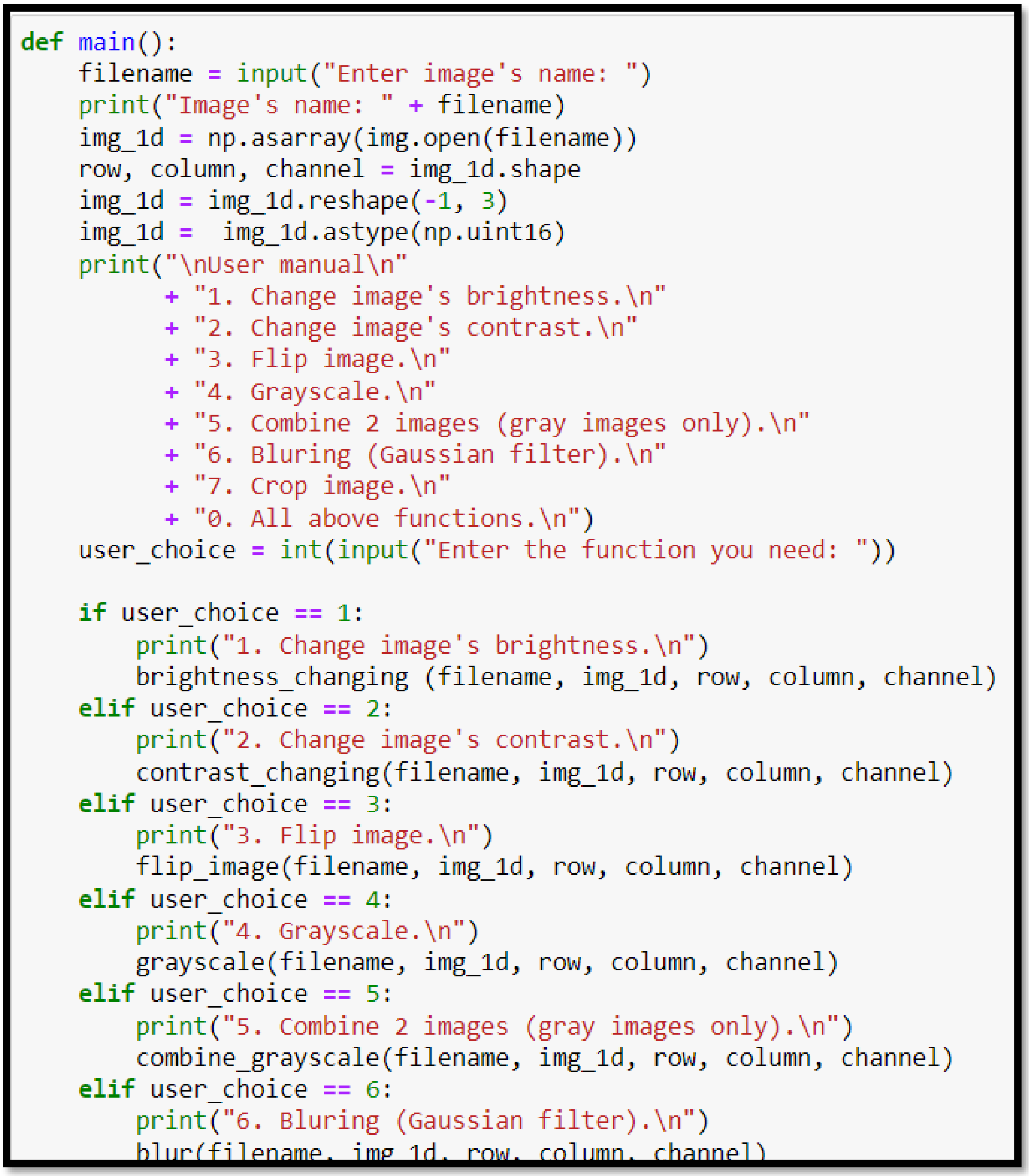
- def cropping(filename, img_1d, row, column, channel):



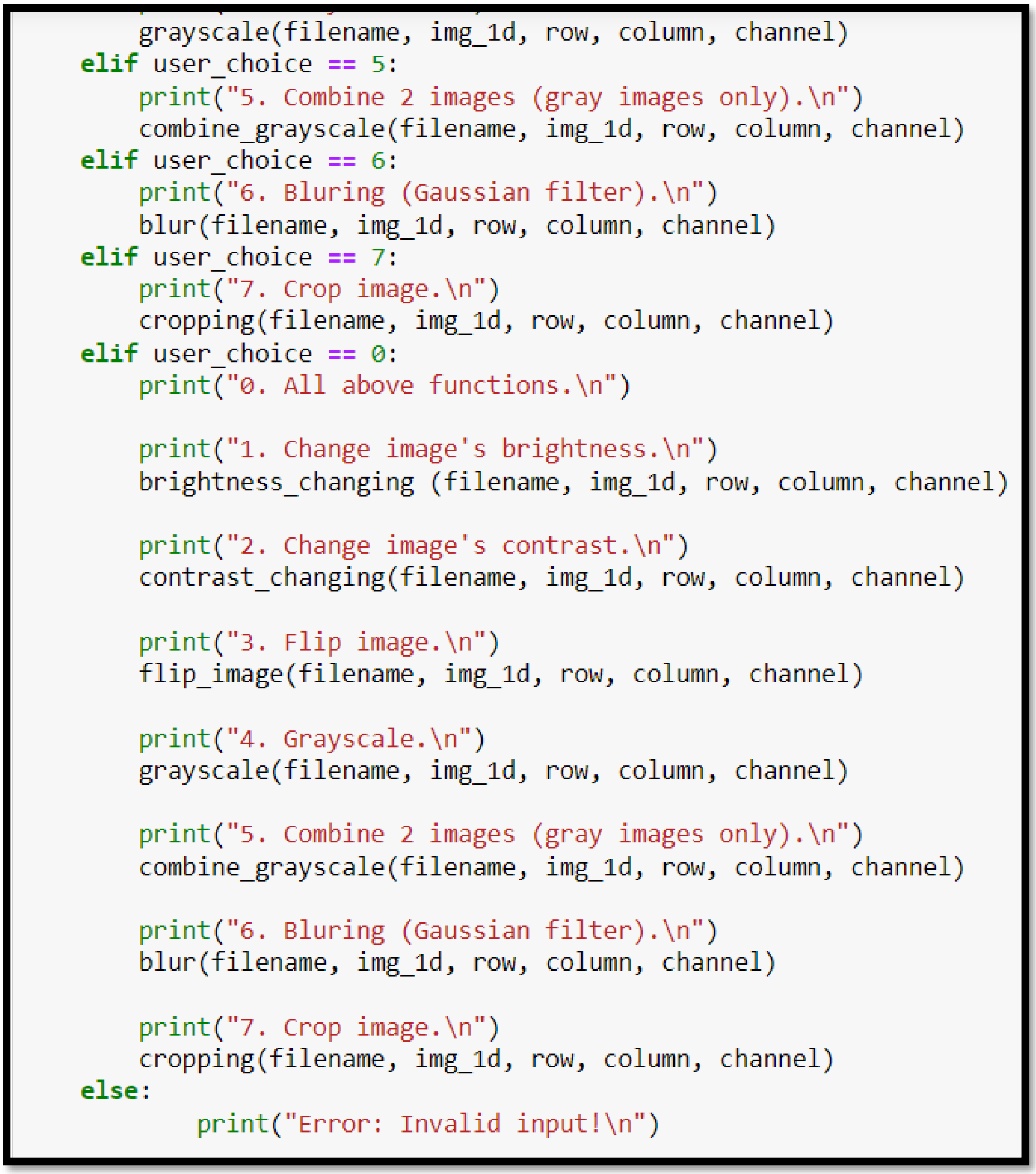
```
def cropping(filename, img_1d, row, column, channel):
    new_image = img_1d.reshape(row, column, channel)
    method = input("Enter the method you want to crop the image"
                   + " ('C' - circular method or 'E' - elliptical method): ")
    method = (method == "C" and "circular") or (method == "E" and "elliptical") or "Error: Invalid input!"
    print("Method: " + method)
    while method == "Error: Invalid input!":
        method = input("Enter the method you want to crop the image"
                       + " ('C' - circular method or 'E' - elliptical method): ")
        method = (method == "C" and "circular") or (method == "E" and "elliptical") or "Error: Invalid input!"
        print("Method: " + method)
    if method == "circular":
        circular_cropping(filename, new_image, row, column)
    else:
        elliptical_cropping(filename, new_image, row, column)
```

- Allow users to enter the method method of cropping image.
- If method = “circular”: call circular_cropping(filename, img_1d, row, column).
- If method = “elliptical”: call elliptical_cropping(filename, img_1d, row, column).

- def main():



```
def main():
    filename = input("Enter image's name: ")
    print("Image's name: " + filename)
    img_1d = np.asarray(img.open(filename))
    row, column, channel = img_1d.shape
    img_1d = img_1d.reshape(-1, 3)
    img_1d =  img_1d.astype(np.uint16)
    print("\nUser manual\n"
          + "1. Change image's brightness.\n"
          + "2. Change image's contrast.\n"
          + "3. Flip image.\n"
          + "4. Grayscale.\n"
          + "5. Combine 2 images (gray images only).\n"
          + "6. Bluring (Gaussian filter).\n"
          + "7. Crop image.\n"
          + "0. All above functions.\n")
    user_choice = int(input("Enter the function you need: "))

    if user_choice == 1:
        print("1. Change image's brightness.\n")
        brightness_changing (filename, img_1d, row, column, channel)
    elif user_choice == 2:
        print("2. Change image's contrast.\n")
        contrast_changing(filename, img_1d, row, column, channel)
    elif user_choice == 3:
        print("3. Flip image.\n")
        flip_image(filename, img_1d, row, column, channel)
    elif user_choice == 4:
        print("4. Grayscale.\n")
        grayscale(filename, img_1d, row, column, channel)
    elif user_choice == 5:
        print("5. Combine 2 images (gray images only).\n")
        combine_grayscale(filename, img_1d, row, column, channel)
    elif user_choice == 6:
        print("6. Bluring (Gaussian filter).\n")
        blur(filename, img_1d, row, column, channel)
```

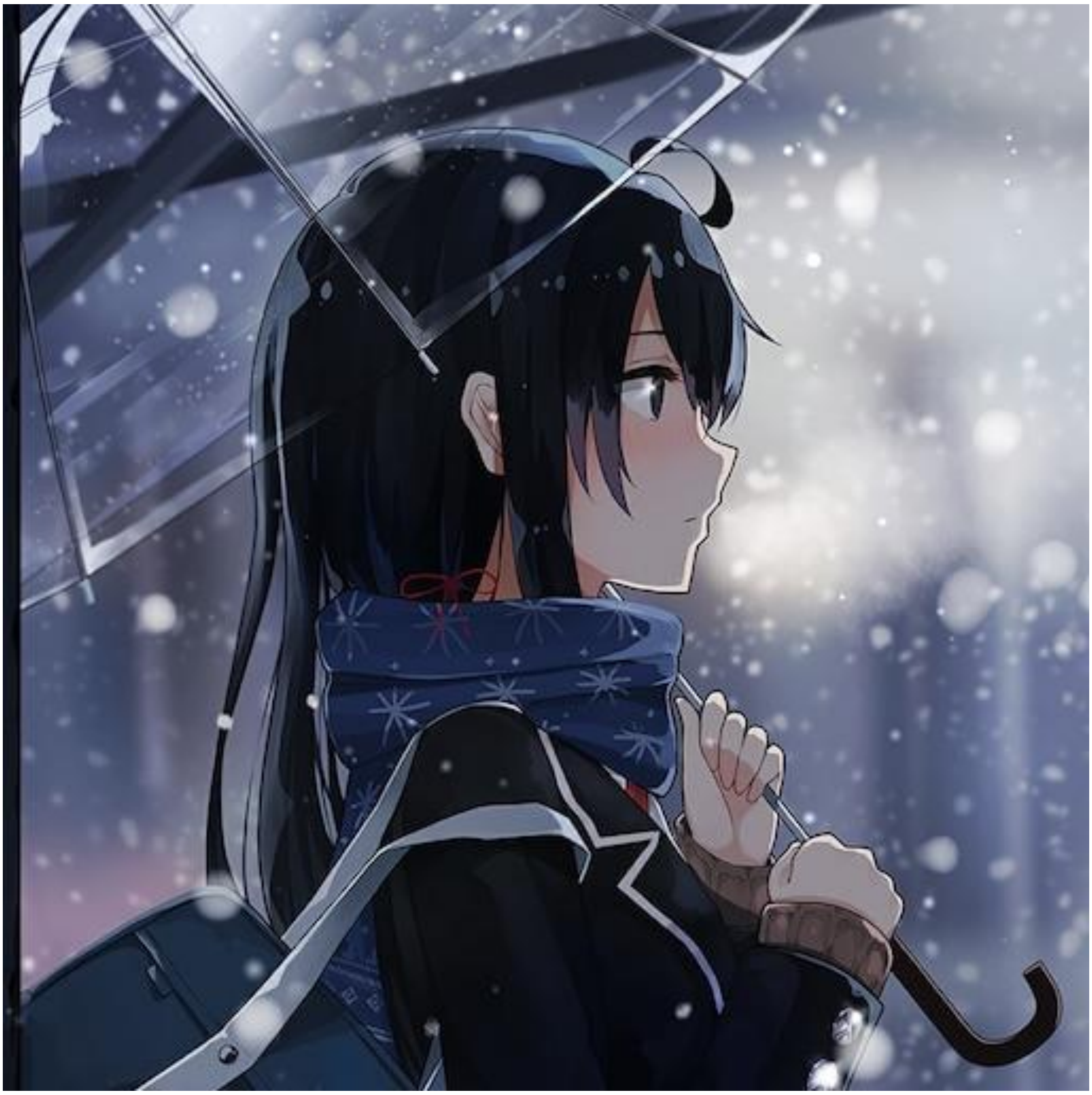


```
        grayscale(filename, img_1d, row, column, channel)
    elif user_choice == 5:
        print("5. Combine 2 images (gray images only).\n")
        combine_grayscale(filename, img_1d, row, column, channel)
    elif user_choice == 6:
        print("6. Bluring (Gaussian filter).\n")
        blur(filename, img_1d, row, column, channel)
    elif user_choice == 7:
        print("7. Crop image.\n")
        cropping(filename, img_1d, row, column, channel)
    elif user_choice == 0:
        print("0. All above functions.\n")

        print("1. Change image's brightness.\n")
        brightness_changing (filename, img_1d, row, column, channel)

        print("2. Change image's contrast.\n")
        contrast_changing(filename, img_1d, row, column, channel)

        print("3. Flip image.\n")
        flip_image(filename, img_1d, row, column, channel)

        print("4. Grayscale.\n")
        grayscale(filename, img_1d, row, column, channel)

        print("5. Combine 2 images (gray images only).\n")
        combine_grayscale(filename, img_1d, row, column, channel)

        print("6. Bluring (Gaussian filter).\n")
        blur(filename, img_1d, row, column, channel)

        print("7. Crop image.\n")
        cropping(filename, img_1d, row, column, channel)
    else:
            print("Error: Invalid input!\n")
```

- Allow users to enter their image’s name filename and proactive in selecting the image processing function user_choice.

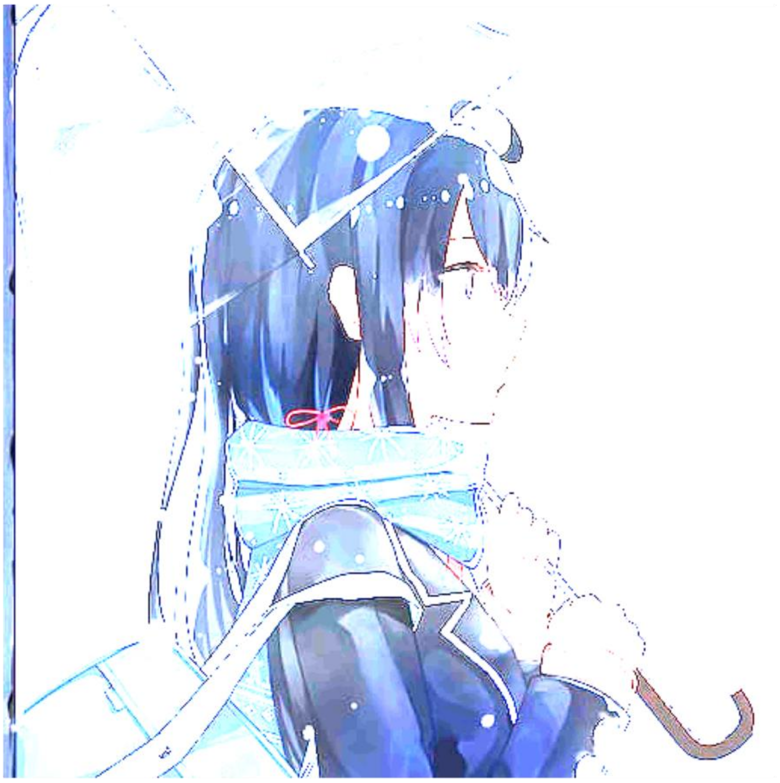
3. Result

- The original iamge:

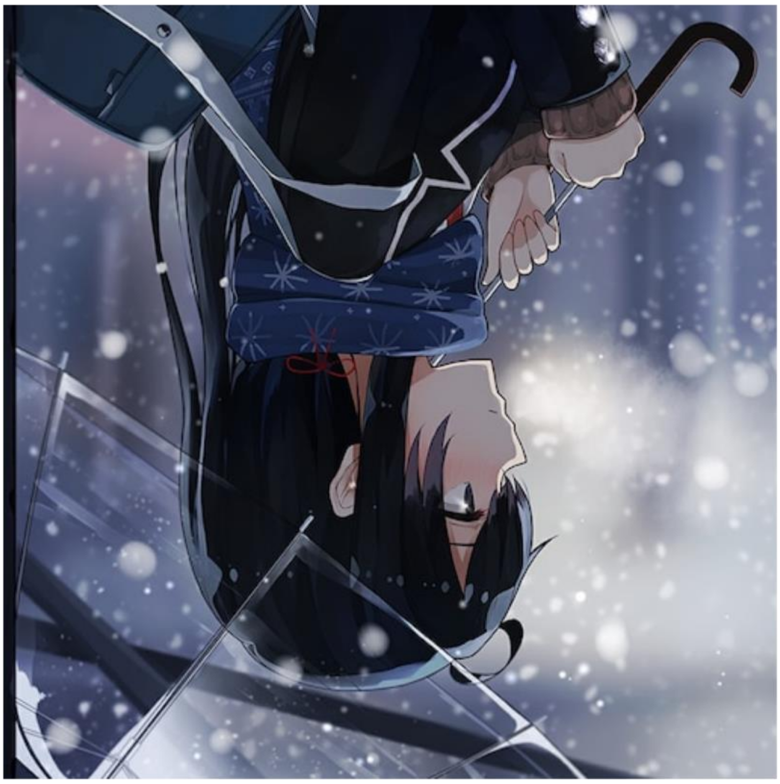


- Change image’s brightness:

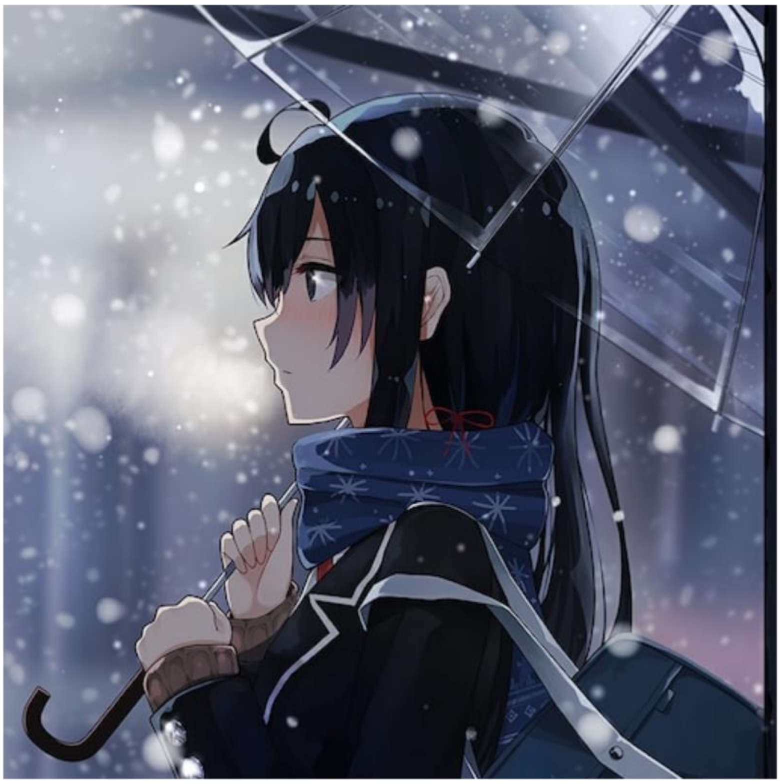
- Change image’s contrast:



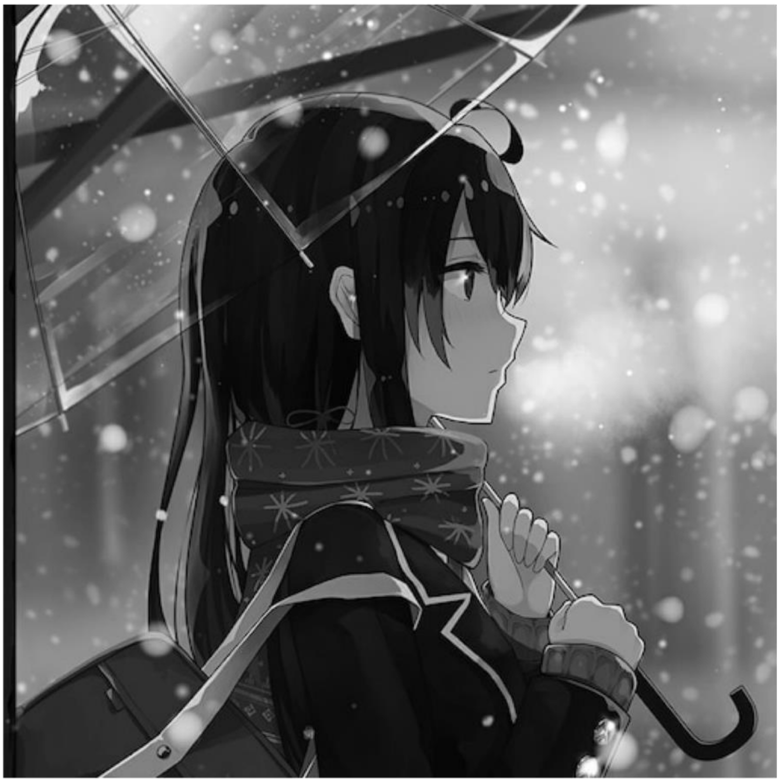
- Flip image vertically and horizontally:
  - ➢ Vertically:



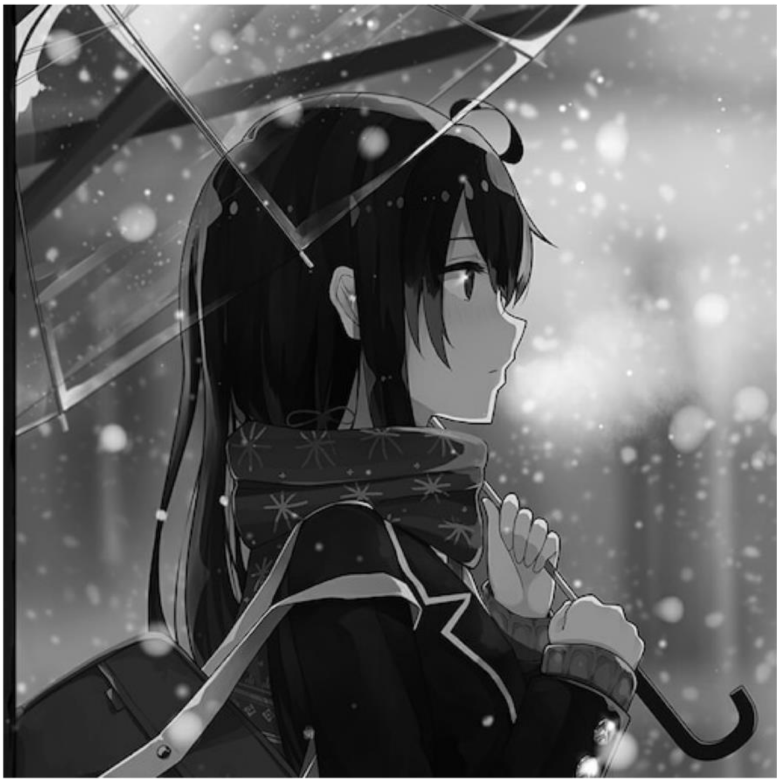
- Horizontally:



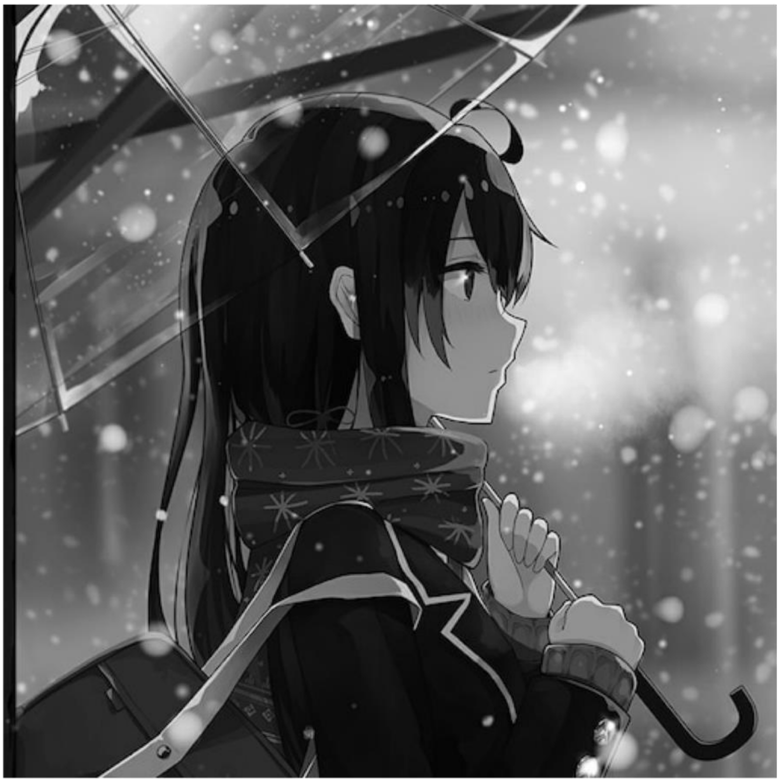
- Convert RGB images to grayscale images:
  - ➢ Average:



- Weighted:

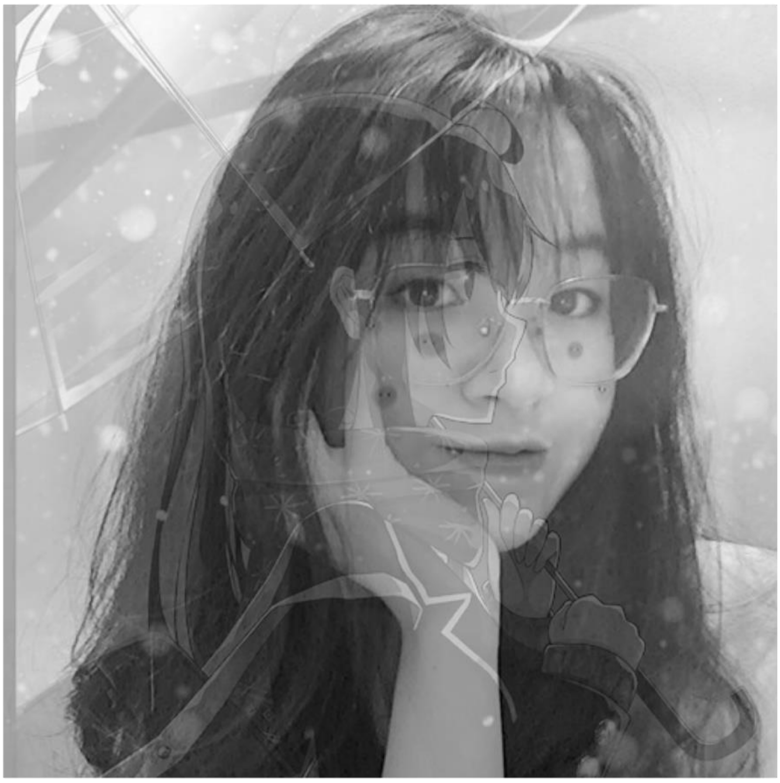


- Combine 2 images of the same size into one:
  - ➢ Fisrt image:

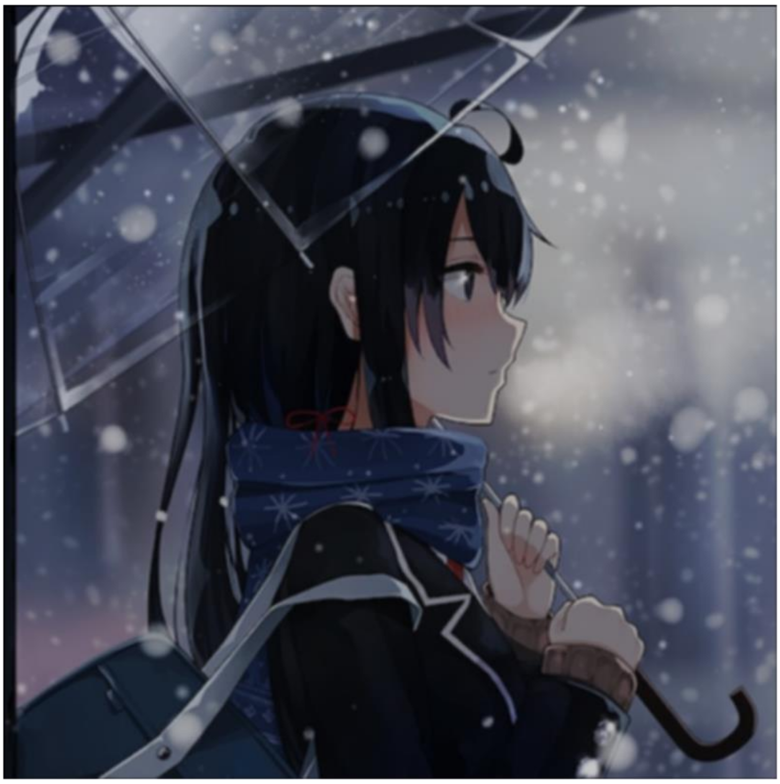


- Second image:

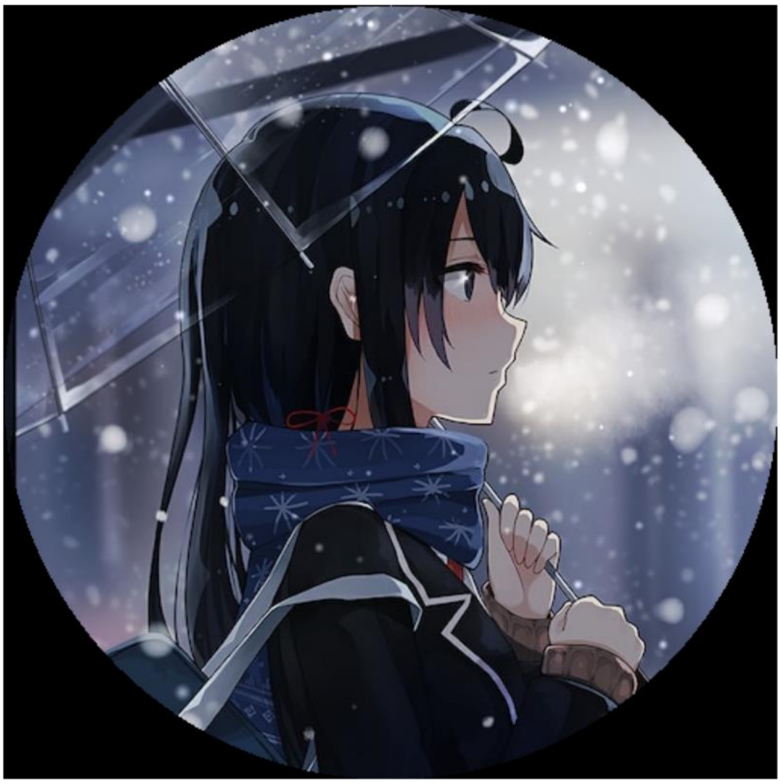
- Result:



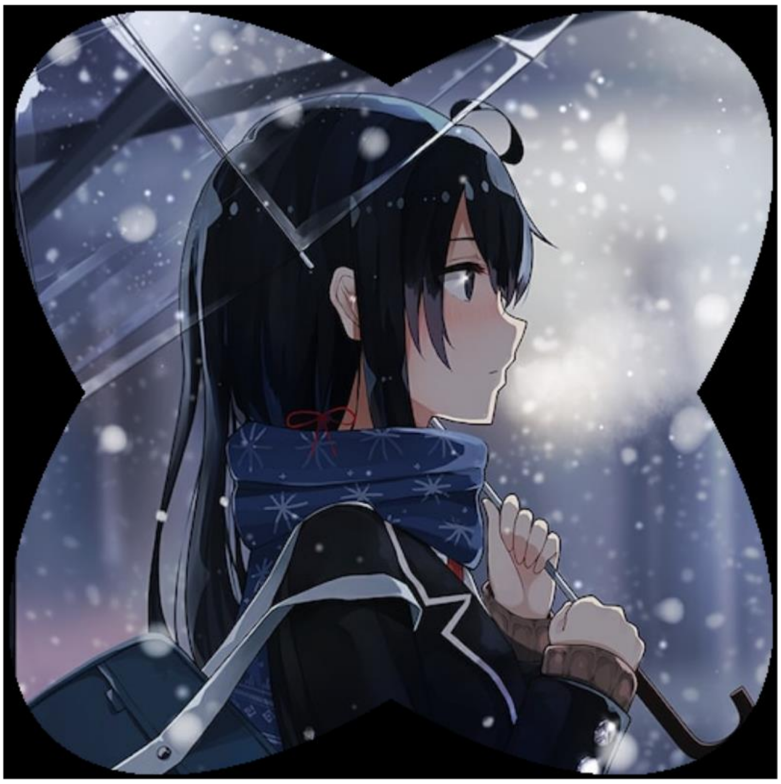
- Blur image:



- Crop image circularly:



- Crop image elliptically:



IV. References

https://numpy.org/doc/stable/index.html

https://pillow.readthedocs.io/en/stable/reference/Image.html?highlight=Image.open#PIL.Image.open

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.savefig.html

https://stackoverflow.com/questions/49142561/change-contrast-in-numpy

https://stackoverflow.com/questions/48406578/adjusting-contrast-of-image-purely-with-numpy

https://datacarpentry.org/image-processing/aio/index.html https://machinetolearn.home.blog/2018/09/16/ml01/

http://minhhn.com/lap-trinh/vector-hinh-hoc-va-cac-khai-niem-ve-vector/#zero-vector-unit-vector

https://viblo.asia/p/nhung-cau-hoi-minh-thuong-hoi-ung-vien-khi-phong-van-ai-phan-1-ly-thuyet-toan-co-ban-ORNZqBxrl0n#_moi-quan-he-giua-cac-dai-luong-tren-5

https://github.com/kvmduc/applied-math/blob/main/Applied%20Math/Lab03/image-processing.py

https://scipy-lectures.org/advanced/image_processing/#examples-for-the-image-processing-chapter

https://stackoverflow.com/questions/55066226/how-to-set-the-minimum-and-maximum-value-for-each-item-in-a-numpy-array

https://note.nkmk.me/en/python-numpy-flip-flipud-fliplr/#:~:text=You%20can%20flip%20the%20image,fliplr()%20.

https://stackoverflow.com/questions/3091316/python-conditional-ternary-operator-for-assignments

https://stackoverflow.com/questions/12201577/how-can-i-convert-an-rgb-image-into-grayscale-in-python

https://stackoverflow.com/questions/41971663/use-numpy-to-convert-rgb-pixel-array-into-grayscale

https://stackoverflow.com/questions/68036122/cv2-grayscale-has-3-channels-and-a-weird-blueish-yellowish-tint

https://stackoverflow.com/questions/69336867/python-error-when-trying-to-multiply-two-floats

https://en.wikipedia.org/wiki/Kernel_(image_processing)

https://datacarpentry.org/image-processing/06-blurring/

https://www.geeksforgeeks.org/apply-a-gauss-filter-to-an-image-with-python/#:~:text=A%20Gaussian%20Filter%20is%20a,to%20get%20the%20desired%20effect.

https://www.youtube.com/watch?v=tXLLpenkM4Y&list=WL&index=3&t=233s&ab_channel=EPSOnlineEducation

https://stackoverflow.com/questions/72048838/combining-various-image-channels-after-gaussian-filtering-produces-white-image

https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/

https://machinelearningcoban.com/2018/10/03/conv2d#-tich-chap-hai-chieu-tong-quat

https://stackoverflow.com/questions/51486297/cropping-an-image-in-a-circular-way-using-python

https://www.pythonfixing.com/2022/01/fixed-how-can-i-create-circular-mask.html

https://math.stackexchange.com/questions/426150/what-is-the-general-equation-of-the-ellipse-that-is-not-in-the-origin-and-rotate

https://realpython.com/python-operators-expressions/