**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

**Subject Name**

**Digital Image Processing**

**Lab Number**

**1**

**SUBMITTED TO:**

**Dr. Asad Khan**

**LE Sundas Ashraf**

**SUBMITTED BY:**

**Student Name**

1. Wahaaj Nasir

**Reg#413238**

**DE- 44 Dept C&SE**

**Objectives:**

Basics of Image Processing in Python

**Related Topic/Chapter in theory class:**

Basics Of Digital Image Processing

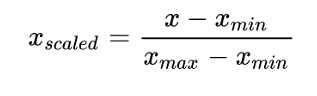
**Hardware/Software required:**

Hardware: PC

Software Tool: Pycharm

**Task 1:**

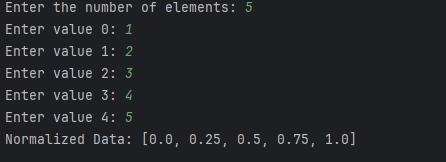
**Write a Python function that takes a list as input and performs the min-max normalization on the list. The function should return the scaled list.**

****

**Solution:**

def normalize\_min\_max(values):  
 highest = max(values)  
 lowest = min(values)  
  
 return [(num - lowest) / (highest - lowest) for num in values]  
  
  
size = int(input("Enter the number of elements: "))  
data = [int(input(f"Enter value {i}: ")) for i in range(size)]  
  
normalized\_data = normalize\_min\_max(data)  
print("Normalized Data:", normalized\_data)

**Output:**

****

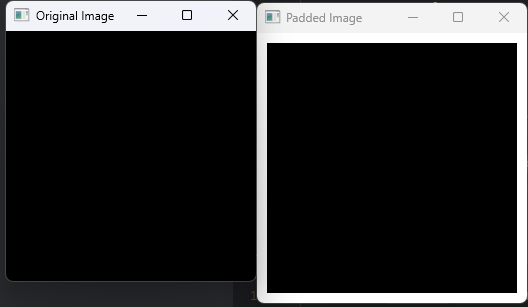
**Task 2:**

**Create an r x c matrix of ones and pad 10 pixels wide border of zeros across each side of it, such that its order will become (5+500+5) x (5+500+5) = 510x510**

**Solution**

import numpy as np  
import cv2 as cv  
  
def apply\_padding(height, width, padding\_size, image):  
 padded\_image = np.full((height + 2 \* padding\_size, width + 2 \* padding\_size), 255, dtype=np.uint8)  
 padded\_image[padding\_size:height + padding\_size, padding\_size:width + padding\_size] = image  
 return padded\_image  
  
height = int(input("Enter the number of rows: "))  
width = int(input("Enter the number of columns: "))  
padding\_size = int(input("Enter padding size: "))  
  
original\_img = np.zeros((height, width), dtype=np.uint8)  
padded\_img = apply\_padding(height, width, padding\_size, original\_img)  
  
cv.imshow("Original Image", original\_img)  
cv.waitKey()  
  
cv.imshow("Padded Image", padded\_img)  
cv.waitKey()  
cv.destroyAllWindows()

**Output:**



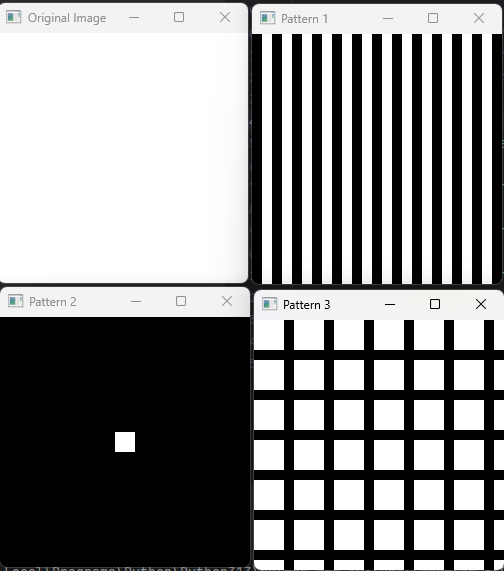
**Task 3**

**Write 3 different Python functions that can create the images given below. Code them in such so that the size of the image itself and the boxes and lines can be changed**

**Solution**

import numpy as np  
import cv2 as cv  
  
def pattern1\_edit(height, width, spacing, image):  
 modified\_image = np.copy(image)  
 for row in range(height):  
 for col in range(0, width, spacing \* 2):  
 modified\_image[row, col:col + spacing] = 0 # Black strip  
 return modified\_image  
  
def pattern2\_edit(height, width, size, image):  
 modified\_image = np.zeros((height, width), dtype=np.uint8) # Start with black image  
 center\_row, center\_col = height // 2, width // 2  
 modified\_image[center\_row - size:center\_row + size, center\_col - size:center\_col + size] = 255 # White square  
 return modified\_image  
  
def pattern3\_edit(height, width, spacing, image):  
 modified\_image = np.copy(image)  
 box\_size = spacing \* 3 # Bigger white box  
  
 # Horizontal black lines  
 for row in range(box\_size, height, box\_size + spacing):  
 modified\_image[row:row + spacing, :] = 0 # Black row  
  
 # Vertical black lines  
 for col in range(box\_size, width, box\_size + spacing):  
 modified\_image[:, col:col + spacing] = 0 # Black column  
  
 return modified\_image  
  
# Input from user  
height = int(input("Enter the number of rows: "))  
width = int(input("Enter the number of columns: "))  
spacing = int(input("Enter spacing size: "))  
  
# Creating a white image  
base\_image = np.full((height, width), 255, dtype=np.uint8)  
  
# Generating modified images  
edited\_img1 = pattern1\_edit(height, width, spacing, base\_image)  
edited\_img2 = pattern2\_edit(height, width, spacing, base\_image)  
edited\_img3 = pattern3\_edit(height, width, spacing, base\_image)  
  
# Display images  
cv.imshow('Original Image', base\_image)  
  
cv.imshow('Pattern 1', edited\_img1)  
  
cv.imshow('Pattern 2', edited\_img2)  
  
cv.imshow('Pattern 3', edited\_img3)  
  
cv.waitKey(0)  
cv.destroyAllWindows()

**Output:**

****

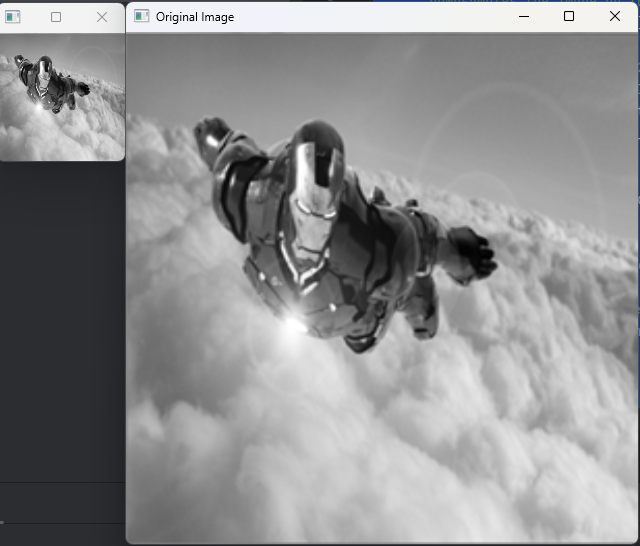
**Task 4**

**Read an image and resize it to 512x512 using the appropriate function. Then down sample the image by 4 so that the final size of the image is 128x128. Display and save the image to the disk**

**Solution**

import numpy as np  
import cv2 as cv  
  
def reduce\_resolution(img, factor=4):  
 *"""Downsamples the image by taking every nth pixel."""* return img[::factor, ::factor]  
  
# Load and resize the image to 512x512  
image\_path = "D:/Uni/Semester 6/DIP/Self/Lab/Lab 1/lab1.png"  
original\_img = cv.imread(image\_path, cv.IMREAD\_GRAYSCALE)  
original\_img = cv.resize(original\_img, (512, 512), interpolation=cv.INTER\_LINEAR)  
  
# Downsample using NumPy slicing  
downsampled\_img = reduce\_resolution(original\_img)  
  
# Save and display images  
cv.imwrite("D:/Uni/Semester 6/DIP/Self/Lab/Lab 1/lab1\_downsample.png", downsampled\_img)  
  
cv.imshow("Original Image", original\_img)  
cv.imshow("Downsampled Image", downsampled\_img)  
  
cv.waitKey(0)  
cv.destroyAllWindows()

**Output**

****

**Task 5**

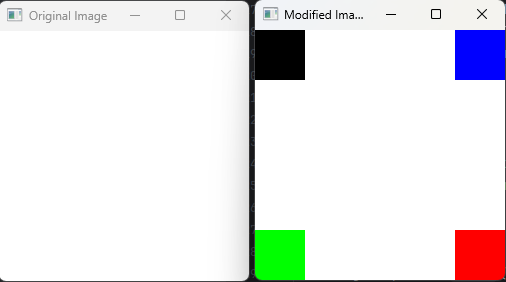
**Write a function to create a white image of 500x500 (or any other size entered by the user) and then create 4 boxes of Red, Green, Blue and Black respectively on each corner of the image as shown below. The size of the colored boxes should be 1/8th the size of the image. (HINT: the arrays of ones and zeros can be in more than 2 dimensions).**

****

**Solution**

import numpy as np  
import cv2 as cv  
  
def draw\_corner\_boxes(image, padding):  
 *"""Adds four colored boxes (black, blue, green, red) at the corners of the image."""* modified\_image = image.copy()  
 height, width = image.shape[:2]  
  
 # Define corner coordinates  
 top\_left = (0, 0, padding, padding) # Black  
 top\_right = (0, width - padding, padding, width) # Blue  
 bottom\_left = (height - padding, 0, height, padding) # Green  
 bottom\_right = (height - padding, width - padding, height, width) # Red  
  
 # Apply colors to corners  
 modified\_image[top\_left[0]:top\_left[2], top\_left[1]:top\_left[3]] = [0, 0, 0] # Black  
 modified\_image[top\_right[0]:top\_right[2], top\_right[1]:top\_right[3]] = [255, 0, 0] # Blue  
 modified\_image[bottom\_left[0]:bottom\_left[2], bottom\_left[1]:bottom\_left[3]] = [0, 255, 0] # Green  
 modified\_image[bottom\_right[0]:bottom\_right[2], bottom\_right[1]:bottom\_right[3]] = [0, 0, 255] # Red  
  
 return modified\_image  
  
# User inputs  
height = int(input("Enter image height: "))  
width = int(input("Enter image width: "))  
padding = int(input("Enter box padding size: "))  
  
# Create white image  
base\_img = np.full((height, width, 3), 255, dtype=np.uint8)  
  
# Add colored boxes  
modified\_img = draw\_corner\_boxes(base\_img, padding)  
  
# Display images  
cv.imshow('Original Image', base\_img)  
cv.imshow('Modified Image', modified\_img)  
  
cv.waitKey(0)  
cv.destroyAllWindows()

**Output**



**Task 6**

**Mirror the image that you have read at center i.e. the lower half of the image should be the copy of the upper half. (HINT: You can use nested loops)**

**Solution**

import numpy as np  
import cv2 as cv  
  
def vertical\_mirror\_flip(image):  
 flipped\_img = image.copy()  
 rows, cols = image.shape  
  
 mid\_row = rows // 2  
  
 # If odd, we take one extra row for bottom half to match the image size  
 if rows % 2 == 1:  
 flipped\_img[mid\_row + 1:, :] = np.flipud(image[:mid\_row, :])  
 else:  
 flipped\_img[mid\_row:, :] = np.flipud(image[:mid\_row, :])  
  
 return flipped\_img  
  
# Load grayscale image  
image\_path = "D:/Uni/Semester 6/DIP/Self/Lab/Lab 1/lab1.png"  
original\_img = cv.imread(image\_path, cv.IMREAD\_GRAYSCALE)  
  
# Display original image  
cv.imshow("Original Image", original\_img)  
cv.waitKey()  
  
# Apply vertical mirror flip  
flipped\_img = vertical\_mirror\_flip(original\_img)  
  
# Display flipped image  
cv.imshow("Flipped Image", flipped\_img)  
cv.waitKey(0)  
cv.destroyAllWindows()

****