**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

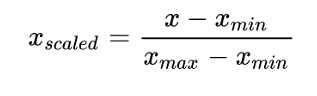
**Hardware/Software required:**

Hardware: PC

Software Tool: Pycharm

**Task 1:**

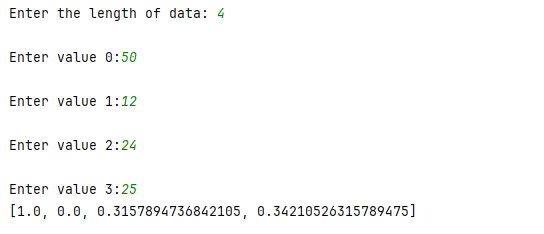
**Write a Python function that takes a list as input and performs the min-max normalization on the list. The function should return the scaled list.**

****

**Solution:**

def min\_max\_normal(my\_list):  
 max\_list = max(my\_list)  
 min\_list = min(my\_list)  
 scaled\_list = []  
 for x in my\_list:  
 x = (x - min\_list)/(max\_list-min\_list)  
  
 scaled\_list.append(x)  
  
 return scaled\_list  
  
length = int(input("Enter the length of data: "))  
data = []  
for x in range(length):  
 temp = int(input("\nEnter value "+ str(x) + ":"))  
 data.append(temp)  
  
normalized = min\_max\_normal(data)  
print(normalized)

**Output:**

****

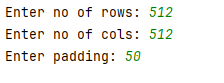
**Task 2:**

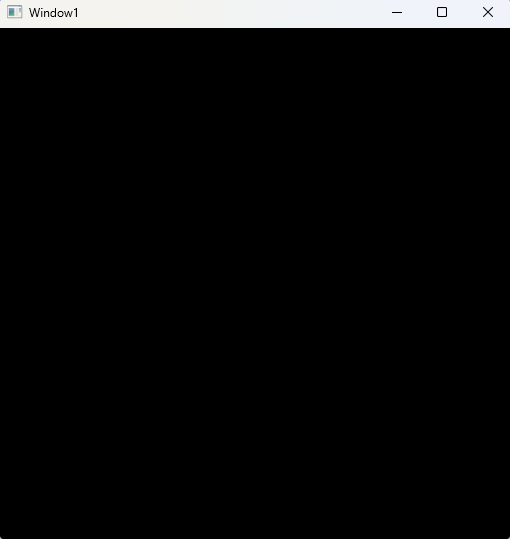
**Create an r x c matrix of ones and pad 10 pixels wide border of zeros across each side of it, such that its order will become (5+500+5) x (5+500+5) = 510x510**

**Solution**

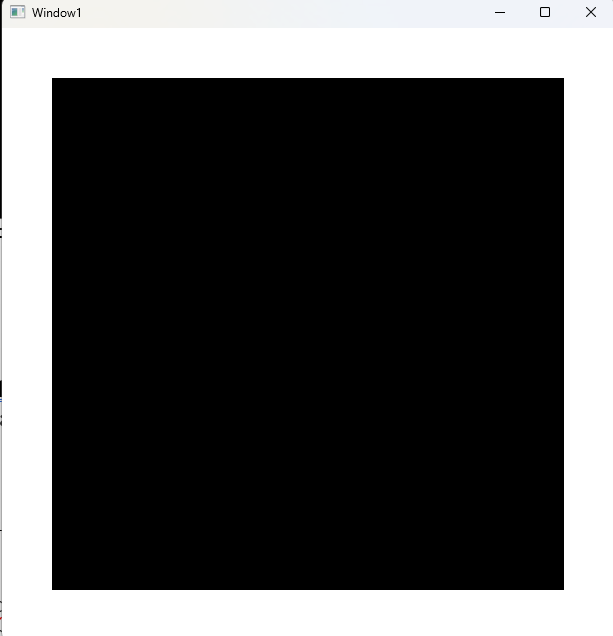
import numpy as np  
import cv2 as cv  
  
def padding(rows, cols, pad, orig):  
 padded\_arr = np.ones((rows+ 2 \* pad, cols+ 2 \* pad), dtype = np.uint8)\*255  
  
 for i in range(rows):  
 for j in range(cols):  
 padded\_arr[i+pad][j+pad] = orig[i][j]  
  
 return padded\_arr  
  
rows = int(input("Enter no of rows: "))  
cols = int(input("Enter no of cols: "))  
pad = int(input("Enter padding: "))  
  
img = np.zeros((rows, cols), dtype = np.uint8)  
pad\_img = padding(rows, cols, pad, img)  
cv.imshow('Window1', img)  
cv.waitKey()  
cv.imshow('Window1', pad\_img)  
cv.waitKey()

**Output:**





**Original Image**



**Image with Border**

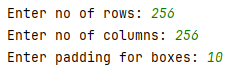
**Task 3**

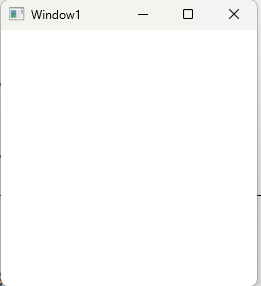
**Write 3 different Python functions that can create the images given below. Code them in such so that the size of the image itself and the boxes and lines can be changed**

**Solution**

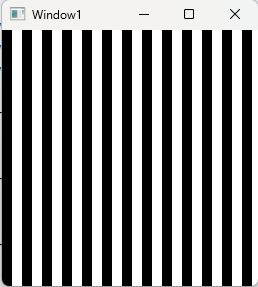
import numpy as np  
import cv2 as cv  
  
def img1\_edit(rows, cols, pad, orig):  
 edit\_img = orig.copy()  
 for i in range(rows):  
 for j in range(0, cols, pad\*2):  
 for k in range(pad):  
 if j + k < cols:  
 edit\_img[i][j+k] = 0  
 return edit\_img  
  
def img2\_edit(rows, cols, pad ,orig):  
 edit\_img = orig.copy() \* 0  
 mid\_row = int(rows/2)  
 mid\_col = int(cols/2)  
  
 top\_left\_r = mid\_row - pad  
 top\_left\_c = mid\_col - pad  
 bottom\_right\_r = mid\_row + pad  
 bottom\_right\_c = mid\_col + pad  
  
 edit\_img[top\_left\_r:bottom\_right\_r, top\_left\_c:bottom\_right\_c] = 255  
  
 return edit\_img  
  
def img3\_edit(rows, cols, pad, orig):  
 edit\_img = orig.copy()  
  
 box\_size = pad \* 3 *# White box should be bigger than the black lines  
  
 # Draw horizontal black lines* for i in range(box\_size, rows, box\_size + pad):  
 for k in range(pad): *# Black line thickness* if i + k < rows:  
 edit\_img[i + k, :] = 0 *# Set entire row to black  
  
 # Draw vertical black lines* for j in range(box\_size, cols, box\_size + pad):  
 for k in range(pad):  
 if j + k < cols:  
 edit\_img[:, j + k] = 0 *# Set entire column to black* return edit\_img  
  
rows = int(input("Enter no of rows: "))  
cols = int(input("Enter no of columns: "))  
pad = int(input("Enter padding for boxes: "))  
orig\_img = np.ones((rows, cols), dtype = np.uint8)\*255 *#This creates white box thingy*orig\_img\_edit\_1 = img1\_edit(rows, cols, pad, orig\_img)  
orig\_img\_edit\_2 = img2\_edit(rows, cols, pad, orig\_img)  
orig\_img\_edit\_3 = img3\_edit(rows, cols, pad, orig\_img)  
  
cv.imshow('Window1',orig\_img)  
cv.waitKey()  
  
cv.imshow('Window1',orig\_img\_edit\_1)  
cv.waitKey()  
  
cv.imshow('Window1',orig\_img\_edit\_2)  
cv.waitKey()  
  
cv.imshow('Window1',orig\_img\_edit\_3)  
cv.waitKey()

**Output:**

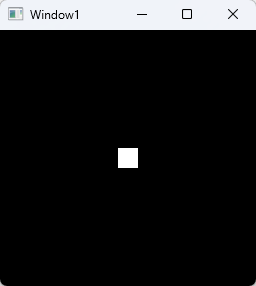
****

****

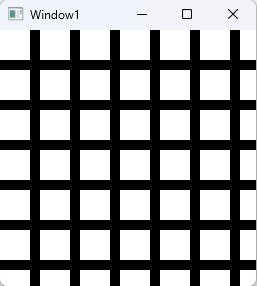
**Original Image**

****

**Edit 1**



**Edit 2**



**Edit 3**

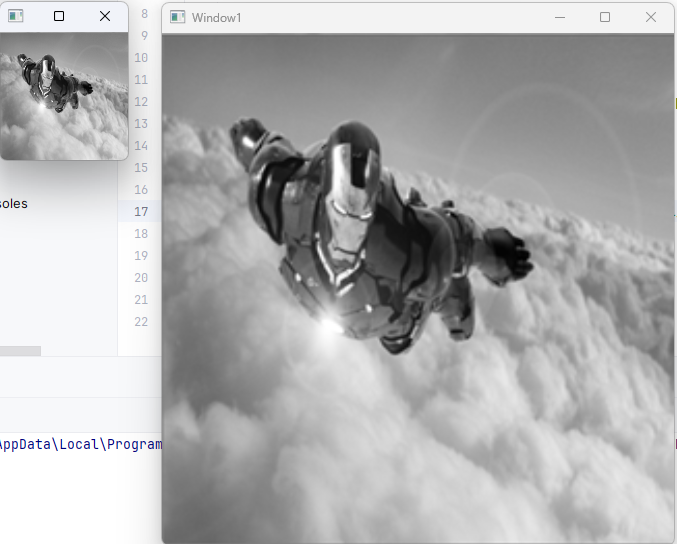
**Task 4**

**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 downsample(image):  
 downsample\_arr = np.ones((128, 128), dtype = np.uint8)  
  
 for i in range(0, 512, 4):  
 for j in range(0,512,4):  
 downsample\_arr[int(i/4)][int(j/4)] = image[i][j]  
 return downsample\_arr  
  
image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lab/Lab 1/lab1.png", 0)  
  
image = cv.resize(image, (512, 512))  
image\_down = downsample(image)  
  
cv.imwrite("D:/Uni/Semester 6/DIP/Self/Lab/Lab 1/lab1\_downsample.png")  
cv.imshow('Window1', image)  
cv.waitKey()  
  
cv.imshow('Window1', image\_down)  
cv.waitKey()

**Output**

****

**Window 1 contains the original image, resized to 512x512. The second window contains the downsampled image.**

****

**Saved the image to Computer as well**

**Task 5**

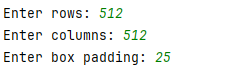
**Write a function to create a white image of 500x500 (or any other size entered by the user) and then create 4 boxes of Red, Green, Blue and Black respectively on each corner of the image as shown below. The size of the colored boxes should be 1/8th the size of the image. (HINT: the arrays of ones and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

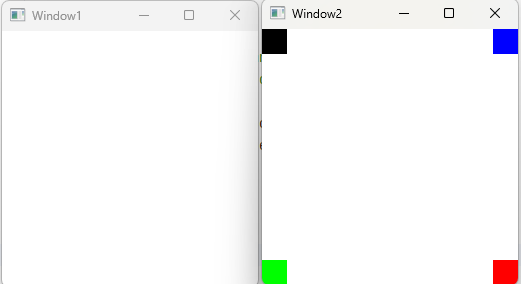
****

**Solution**

import numpy as np  
import cv2 as cv  
  
def add\_boxes(image, pad):  
 edit\_image = orig\_image.copy()  
 rows, cols = image.shape[:2]  
 *#Black* edit\_image[0:pad, 0:pad] = [0, 0 , 0]  
  
 *#Blue* edit\_image[0:pad,cols-pad:cols] = [255, 0 ,0]  
  
 *#Green* edit\_image[rows-pad:rows, 0:pad] = [0, 255, 0]  
  
 *#Red* edit\_image[rows-pad:rows, cols-pad:cols] = [0, 0 , 255]  
  
 return edit\_image  
rows = int(input("Enter rows: "))  
cols = int(input("Enter columns: "))  
pad = int(input("Enter box padding: "))  
  
orig\_image = np.ones((rows, cols, 3), dtype = np.uint8)\*255  
box\_img = add\_boxes(orig\_image, pad)  
  
cv.imshow('Window1', orig\_image)  
cv.waitKey()  
  
cv.imshow('Window1', box\_img)  
cv.waitKey()

**Output**





**Original and Edited**

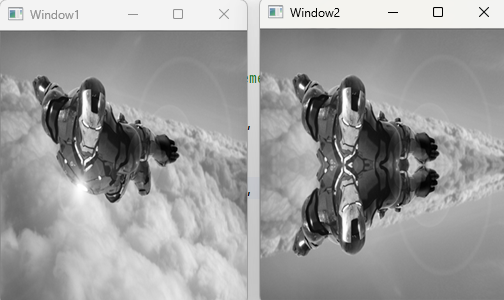
**Task 6**

**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 flip(image):  
 flipped\_img = image.copy()  
 rows, cols = image.shape  
  
 mid\_rows = rows // 2  
 flipped\_temp = np.ones((mid\_rows, cols),dtype = np.uint8)  
  
 for i in range(mid\_rows):  
 for j in range(cols):  
 flipped\_temp[i][j] = image[i][j]  
  
 flipped\_temp = flipped\_temp[::-1, :]  
  
 for i in range(mid\_rows):  
 for j in range(cols):  
 flipped\_img[i+mid\_rows][j] = flipped\_temp[i][j]  
  
 return flipped\_img  
  
image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lab/Lab 1/lab1.png", 0)  
  
cv.imshow('Window1', image)  
cv.waitKey()  
flipped\_img = flip(image)  
cv.imshow('Window1', flipped\_img)  
cv.waitKey()

**Output:**

****

**The Original image and then the flipped version**