**DIGITAL IMAGE PROCESSING COURSE - 505060   
PRACTICE LABS**

**LAB 03. BINARY IMAGE PROCESSING**

Requirements

1. Follow the instructions with the help from your instructor.
2. Finish all the exercises in class and do the homework at home. You can update your solutions after class and re-submit all your work together with the homework.
3. Grading  
   Total score = 50% \* Attendance + 50% \* Exercises

Rules:

* If the number of finished exercises is less than **80% total number of excercises**, you will get **zero** for the lab.
* Name a source file as “**src\_XX.py**” where XX is the exercise number, for ex., “src\_03.py” is the source code for the Exercise 3.
* Add the text of your Student ID to each of the output image.
* Name an output image as “**image\_XX\_YY.png**” where XX is the exercise number and YY is the order of output images in the exercise, for ex., “image\_03\_02.png” is the second output image in the Exercise 3.
* Submit the source code and output image files directly to Google classroom assignment, donot compress the files.

If you submit the exercises with wrong rules, you will get **zero** for the lab or the corresponding exercises.

1. Plagiarism check

If any 2 of the students have the same output images, then all will get zero for the corresponding exercises.

**INTRODUCTION**

In this Lab, you will learn how to

* Bitwise Operations
* Masking of images
* Alpha Blending with OpenCV
* Image thresholding

**INSTRUCTIONS**

1. **Bitwise Operations**

[OpenCV: Operations on arrays: ***bitwise\_and()***](https://docs.opencv.org/4.0.1/d2/de8/group__core__array.html#ga60b4d04b251ba5eb1392c34425497e14)

dst **=** cv2**.**bitwise\_and(src1, src2[, dst[, mask]])

cv2.bitwise\_and() is a function that performs bitwise AND processing as the name suggests. The AND of the values for each pixel of the input images src1 and src2 is the pixel value of the output image.

Other [Bitwise operation in Python (OR, XOR, NOT, SHIFT)](https://note.nkmk.me/en/python-bit-operation/)

dst **=** cv2**.**bitwise\_or(src1, src2[, dst[, mask]])

dst **=** cv2**.**bitwise\_xor(src1, src2[, dst[, mask]])

dst **=** cv2.bitwise\_not(src[, dst[, mask]])

Source: <https://www.digitalocean.com/community/tutorials/arithmetic-bitwise-and-masking-python-opencv>

Let’s create a bitwise square and a bitwise circle through which we can use the bitwise operations.

# creating a rectangle

rectangle = np.zeros((300, 300), dtype="uint8")

cv2.rectangle(rectangle, (25, 25), (275, 275), 255, -1)

cv2.imshow("Rectangle : ", rectangle)

# creating a circle

circle = np.zeros((300, 300), dtype="uint8")

cv2.circle(circle, (150, 150), 150, 255, -1)

cv2.imshow("Circle : ", circle)

# the bitwise\_and function executes the AND operation

# on both the images

bitwiseAnd = cv2.bitwise\_and(rectangle, circle)

cv2.imshow("AND", bitwiseAnd)

cv2.waitKey(0)

# the bitwise\_or function executes the OR operation

# on both the images

bitwiseOr = cv2.bitwise\_or(rectangle, circle)

cv2.imshow("OR", bitwiseOr)

cv2.waitKey(0)

# the bitwise\_xor function executes the XOR operation

# on both the images

bitwiseXor = cv2.bitwise\_xor(rectangle, circle)

cv2.imshow("XOR", bitwiseXor)

cv2.waitKey(0)

# the bitwise\_not function executes the NOT operation

# on both the images

bitwiseNot = cv2.bitwise\_not(rectangle, circle)

cv2.imshow("NOT", bitwiseNot)

cv2.waitKey(0)

1. **Masking of images**

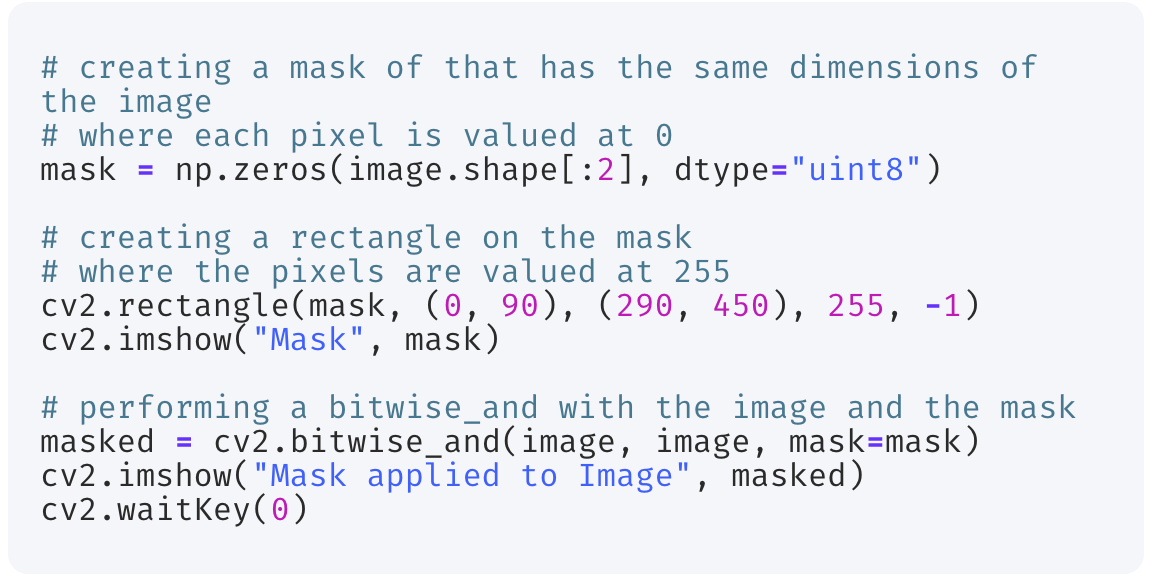
Masking is used in Image Processing to output the Region of Interest, or simply the part of the image that we are interested in. We tend to use bitwise operations for masking as it allows us to discard the parts of the image that we do not need.

We have three steps in masking.

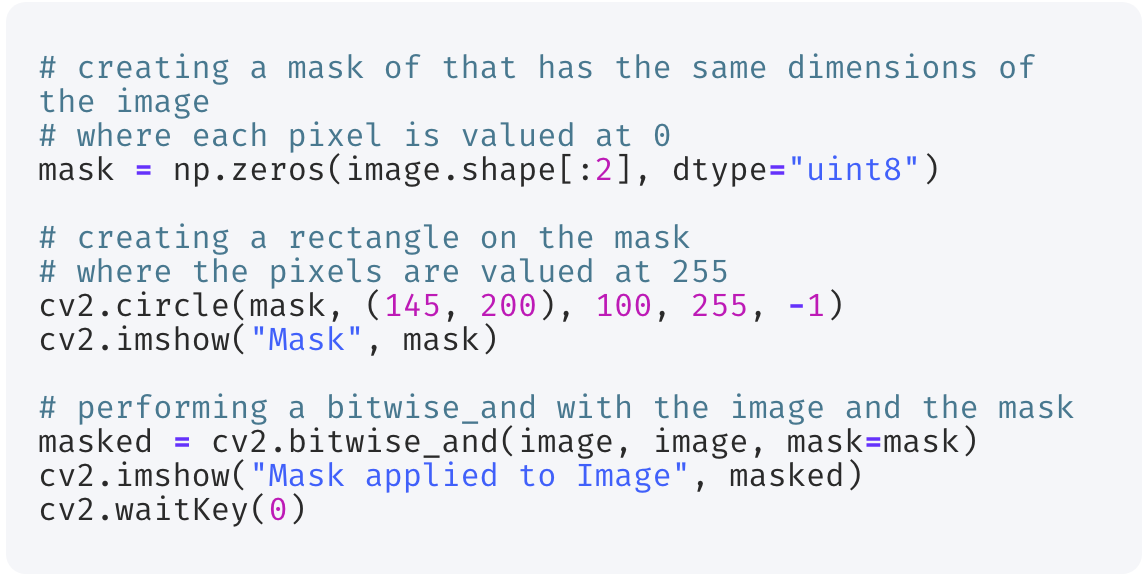
1. Creating a **black** canvas with ***the same dimensions*** as the image, and naming it as mask.
2. Changing the values of the mask by drawing any figure in the image and providing it with a **white** color.
3. Performing the **bitwise AND** operation on the image with the mask.

Example:

Masking with a rectangle mask.



Masking with a circle mask.



1. **Alpha Blending with OpenCV**

Use cv2.addWeighted() to do alpha blending with OpenCV.

dst **=** cv2**.addWeighted**(src1, alpha, src2, beta, gamma[, dst[, dtype]])

It is calculated as follows according to parameters. The fifth parameter gamma is the value to be added to all pixel values.

dst **=** src1 **\*** alpha **+** src2 **\*** beta **+** gamma

The two images need to be the same size, so resize them.

import cv2

src1 **=** cv2**.**imread('[lena.jpg](https://raw.githubusercontent.com/nkmk/python-snippets/817ec63c20dd1e3289327948c943abe646fe10d4/notebook/data/src/lena.jpg)')

src2 **=** cv2**.**imread('[rocket.jpg](https://raw.githubusercontent.com/nkmk/python-snippets/817ec63c20dd1e3289327948c943abe646fe10d4/notebook/data/src/rocket.jpg)')

src2 **=** cv2**.**resize(src2, src1**.**shape[1::**-**1])

The image is alpha blended according to the values of the second parameter alpha and the fourth parameter beta. Although images are saved as files here, if you want to display them in another window, you can use cv2.imshow() (eg: cv2.imshow('window\_name', dst)). The same is true for the following sample code.

dst **=** cv2**.**addWeighted(src1, 0.5, src2, 0.5, 0)

cv2**.**imwrite('opencv\_add\_weighted.jpg', dst)



1. **Image thresholding**

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. Image thresholding is most effective in images with high levels of contrast.

The basic Thresholding technique is Binary Thresholding. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value. The different Simple Thresholding Techniques are:

th, dst = cv2.**threshold**(source, thresholdValue, maxVal, *thresholdingTechnique*)

## cv2.*THRESH\_BINARY* If pixel intensity is greater than the set threshold, value set to *maxval*, else set to 0 (black)

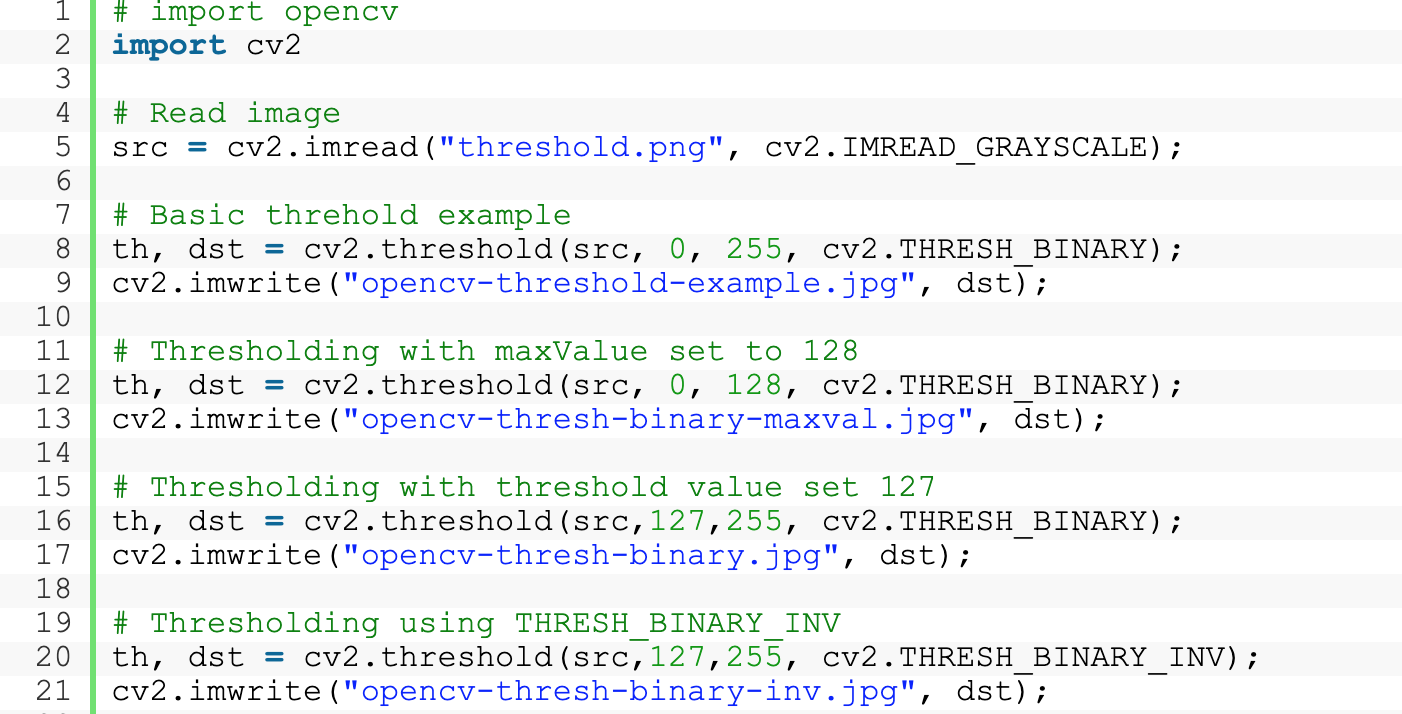
## cv2.*THRESH\_BINARY\_INV* Inverted case of cv2.THRESH\_BINARY

## cv.*THRESH\_TRUNC* If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be the same as the threshold. All other values remain the same.

## cv.*THRESH\_TOZERO* Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.

## cv.*THRESH\_TOZERO\_INV* Inverted case of cv2.THRESH\_TOZERO.

## 

****Source: <https://learnopencv.com/opencv-threshold-python-cpp/>

**EXERCISE**

**Ex1. Masking image**

Extracting the faces from the input image using a circular mask.

Output is 3 of face images extracted from the input image.

***Hint***: Using circular masks as in the instructions above.

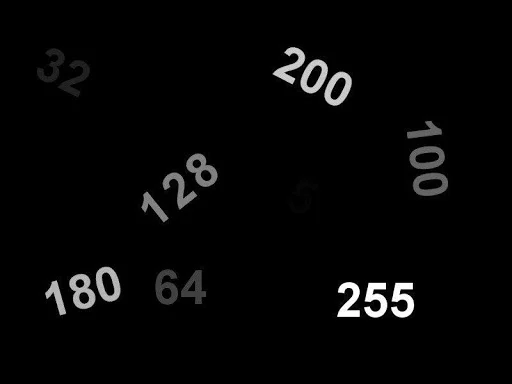


**Ex2. Blend the following 2 images to create a new image**

**Ex3. Image thresholding**

Given an input image as follows (numbers are pixel intensities)



1. Convert the image into 02 binary images:
   1. 1st output : numbers greater than or equals to 180 are in black
   2. 2nd output: numbers less than 180 are in white  
      **Hints**: Using different thresholds and binarization techniques.
2. Extract each number in the input image as in separated images.

**HOMEWORK**

Create a video from your webcam by inserting the TDT logo to images streaming from your webcam.