SESSION 3

1. NumPy Topics - Copy & View
2. Reshaping & Flattening the NumPy arrays
3. Sorting the NumPy Arrays
4. Introduction to openCV-Python
5. OpenCV Topics - vstack, hstack, indexing, cropping, slicing,filtering:
   5.1 OpenCV and NumPy Operations: vstack
   5.2 OpenCV and NumPy Operations: hstack
   5.3 OpenCV and NumPy Operations: indexing
   5.4 OpenCV and NumPy Operations: Crop using indexing
   5.5 OpenCV and NumPy Operations: Cropping Numpy Zeros
   5.6 OpenCV and NumPy Operations: Extracting Channel using indexing
   5.7 OpenCV and NumPy Operations: Cropping & Patching
   5.7 OpenCV and NumPy Operations: Gray Slicing
   5.8 OpenCV and NumPy Operations: Masking using Filtering

1. NumPy Topics - Copy & View

- In NumPy, there are two main ways to create a copy of an array: `"shallow copy"` and `"deep copy"`.
- A shallow copy is created using the `view()` method or by simply assigning the array to a new variable.
- A deep copy, on the other hand, creates a new array object with its own data. You can create a deep copy using the `copy()` method.

Copy:
The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not be affected to the copy.

View:
The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original will affect the view.

NumPy Array View

- A view of an array is a new array object that shares the same data with the original array.
- Any changes made to the view will also affect the original array.
- In NumPy, you can create a view of an array using the `view()` method.

In [ ]:
```
#Example of numPy View creation
import numpy as np

a = np.array([1, 2, 3])
b = a.view()

b[0] = 0

print(a)
```

```
[0 2 3]
```

NumPy Array Copy

- A copy of an array is a new array with its own data, and changes made to the copy do not affect the original array.
- In NumPy, you can create a copy of an array using the `copy()` method.

In [ ]:
```
#Example of numPy Copy creation
import numpy as np

a = np.array([1, 2, 3])
b = a.copy()

b[0] = 0
# In this case, changing the first element of b did not affect the first element of a.
print(a)
```

```
[1 2 3]
```

Example Code to peform Array Checks:

Check if an Array Owns the data that it's copy owns, and the view does not own the original array's data.

Every NumPy array has the attribute base that returns None if the array owns the data.

Otherwise, the base attribute refers to the original object.

In [ ]:
```
#Print the value of the base attribute to check if an array owns it's data or not:

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

x = arr.copy()
y = arr.view()

print('NumPy Copy Output: ',x.base) # as x is a copy & it owns the data it holds
print('NumPy View Output: ',y.base) # as y is a view of arr & it doesn't owns the data it holds
```

```
NumPy Copy Output:  None
NumPy View Output:  [1 2 3 4 5]
```

> *1-dimensional NumPy Array as highlighted in the below image*

Copying and changing the array

In [ ]:
```
# Examples of NumPy Copy & Change
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
# copying the original array
x = arr.copy()
# changing the original array
arr[0] = 42

print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

Make Changes in the View

In [ ]:

```
# Make a view, change the view, and display both arrays:

import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
print(arr)
x[0] = 31

print(x)
```

```
[1 2 3 4 5]
[31  2  3  4  5]
```

NumPy Array : Copy vs View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

In [ ]:
```
## COPY:
## Make a copy, change the original array, and display both arrays:

import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

In [ ]:
```
# VIEW:
# Make a view, change the original array, and display both arrays:

import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
print(arr)
arr[0] = 42


print(x)
```

```
[1 2 3 4 5]
[42  2  3  4  5]
```

---

2. Reshaping & Flattening the NumPy arrays

- **Reshaping an array refers to changing its shape or dimensions without changing its total number of elements.**
- **Flattening array means converting a multidimensional array into a 1D array.**

In [ ]:
```
# Convert 1D array with 8 elements to 3D array with 2x2 elements:

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(2, 2, -1)

print(newarr)
```

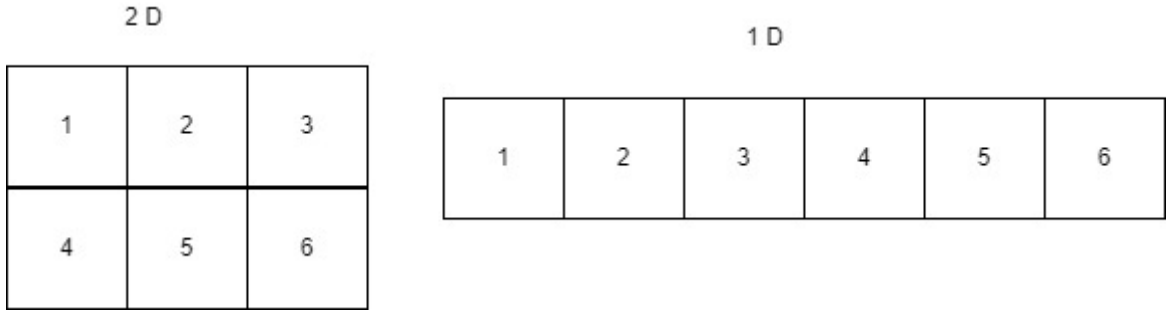
```
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

Flattening array means converting a multidimensional array into a 1D array.

We can use reshape(-1) to do this.

In [ ]:
```
# Convert the array into a 1D array:

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

newarr = arr.reshape(-1)

print(newarr)
```

```
[1 2 3 4 5 6]
```



---

3. Sorting the NumPy Arrays

Sorting means putting elements in an ordered sequence.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

In [ ]:
```
#Sort the array:

import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

```
[0 1 2 3]
```

In [ ]:
```
# Sort the array alphabetically:

import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

```
['apple' 'banana' 'cherry']
```

Sorting a 2-D Array

In [ ]:
```
# If you use the sort() method on a 2-D array, all the rows of that 2-D array will be sorted:

# Sort a 2-D array :

import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

```
[[2 3 4]
 [0 1 5]]
```

---

4. Introduction to openCV-Python

openCV was initailly developed in c++ a very fast compiled language as compared to the slow nature of python. But due to the simplicity of python and the ease of use, a wrapper was developed for python around the original openCV c++ code. Which allowed the users to make the best of both world. A wrapper in simple terms means that the user will be writing the code in python but in the backend the code will be executed in c++ which made the code execution faster.

Installing openCV

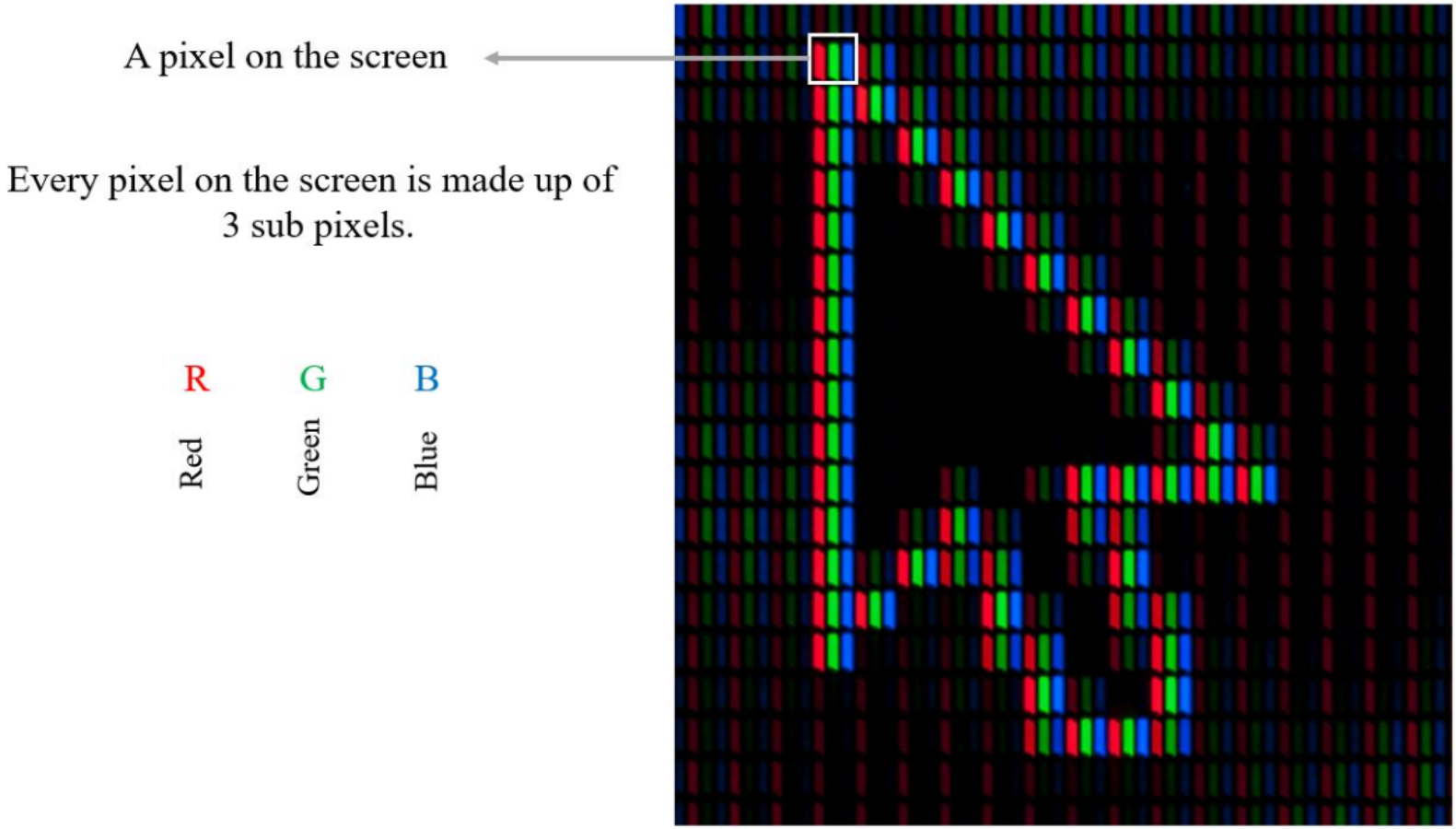
- for windows users
  ```
  pip install opencv-python
  ```

- for mac users
  ```
  pip3 install opencv-python
  ```

Understanding Images

Before we start with image processing we must understand what an image is.

Below Image shows a zoomed in image of a mouse pointer with white edges on the screen.



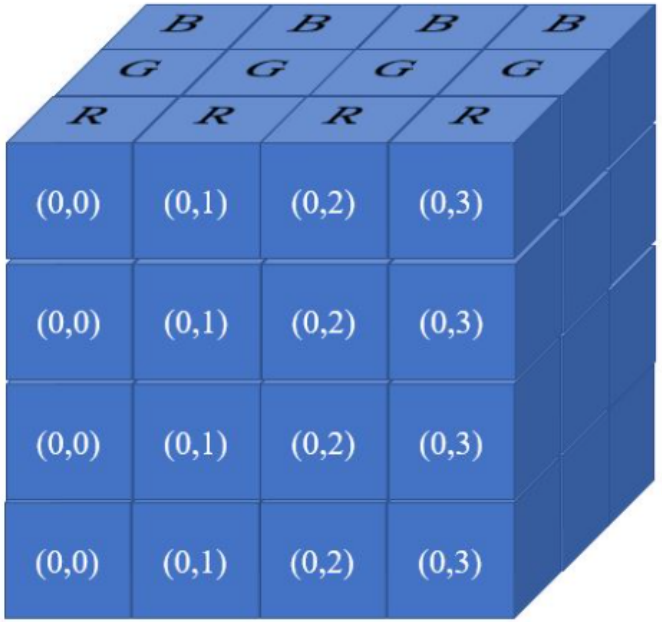
Wait but the edges of the mouse is not white!!

The white color is the result of the RGB sub-pixel colors mixing together as they are close to each other. Red Green and Blue being the primary colors we can create any color on the screen by manipulating the brightness of each sub-pixel.

- A standard monitor supports 256 levels of brightness for every sub-pixel. So by manipulating the RGB sub-pixels we can create a total number of 256^3 colors.
- A subpixel brightness of 0 denotes that the subpixel is entirely turned off while a value of 255 denotes the subpixel is completely turned on.

You can use the below link to check out a RGB color picker which allows you to chose the RGB brightness to create a new color.
https://programmingdesignsystems.com/color/color-models-and-color-spaces/index.html

An image on the screen can be represented using a 3D numpy array having the number of rows and columns equal to the height and the width of the image and the third dimension representing the RGB subpixel values



---

5. OpenCV Topics - vstack, hstack, indexing, cropping, slicing,filtering.

5.1 OpenCV and NumPy Operations: **vstack**

To use OpenCV's vstack function to stack two images vertically using NumPy, you can follow these steps:

1. Import the necessary libraries
2. Load your images using `cv2.imread`
3. Resize the images to have the same width
4. Stack the images vertically using `np.vstack`
5. Display the stacked image using `cv2.imshow`

In [ ]:
```
# OpenCV's vstack function :

import cv2
import numpy as np

img1 = cv2.imread('Images/image1.png')
img2 = cv2.imread('Images/image2.png')

height, width = img1.shape[:2]
img2 = cv2.resize(img2, (width, height))

stacked_img = np.vstack((img1, img2))

cv2.imshow('Stacked Image', stacked_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5.2 OpenCV and NumPy Operations: **hstack**

- To use OpenCV's hstack function to stack two images horizontally using NumPy.

In [ ]:
```
# OpenCV's hstack function
import cv2
import numpy as np

img1 = cv2.imread('Images/image1.png')
img2 = cv2.imread('Images/image2.png')
```

```
height, width = img1.shape[:2]
img2 = cv2.resize(img2, (width, height))

stacked_img = np.hstack((img1, img2))

cv2.imshow('Stacked Image', stacked_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

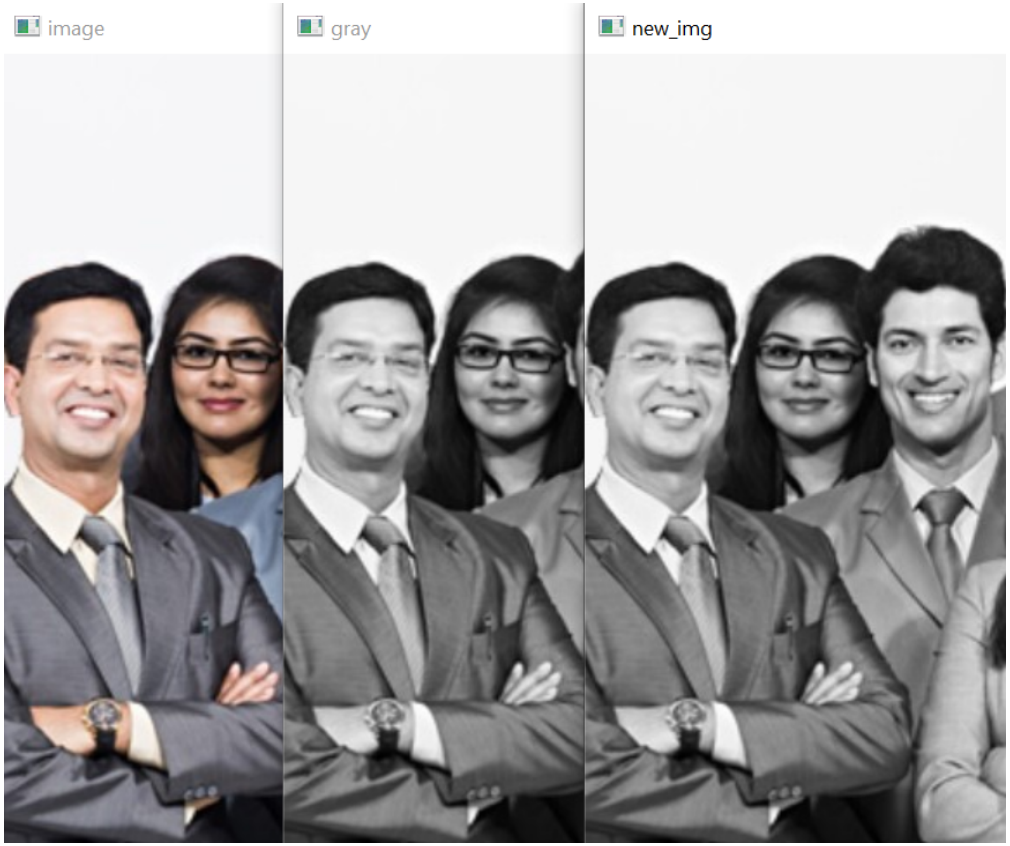
5.3 OpenCV and NumPy Operations: **indexing**

- We first load an image using OpenCV and display it.
- Then, we convert the image to grayscale using OpenCV's cvtColor function.
- We then get the dimensions of the grayscale image and create a new NumPy array of zeros with the same dimensions.
- We then loop through each pixel in the grayscale image and set the corresponding pixel in the new NumPy array.
- Finally, we display and save the new image.

In [ ]:

```
import cv2
import numpy as np

# Load an image using OpenCV
img = cv2.imread('Images\\example.jpg')

# Display the image
cv2.imshow('image', img)
##cv2.waitKey(0)

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
cv2.imshow('gray', gray)
##cv2.waitKey(0)
```

```
# Get the dimensions of the grayscale image
height, width = gray.shape

# Create a new NumPy array of zeros with the same dimensions as the grayscale image
new_img = np.zeros((height, width), dtype=np.uint8)

# Loop through each pixel in the grayscale image and set the corresponding pixel in the new image
for i in range(height):
    for j in range(width):
        new_img[i][j] = gray[i][j]

# Display the new image
cv2.imshow('new_img', new_img)
cv2.waitKey(0)

# Save the new image
cv2.imwrite('new_img.jpg', new_img)

# Release all windows
cv2.destroyAllWindows()
```



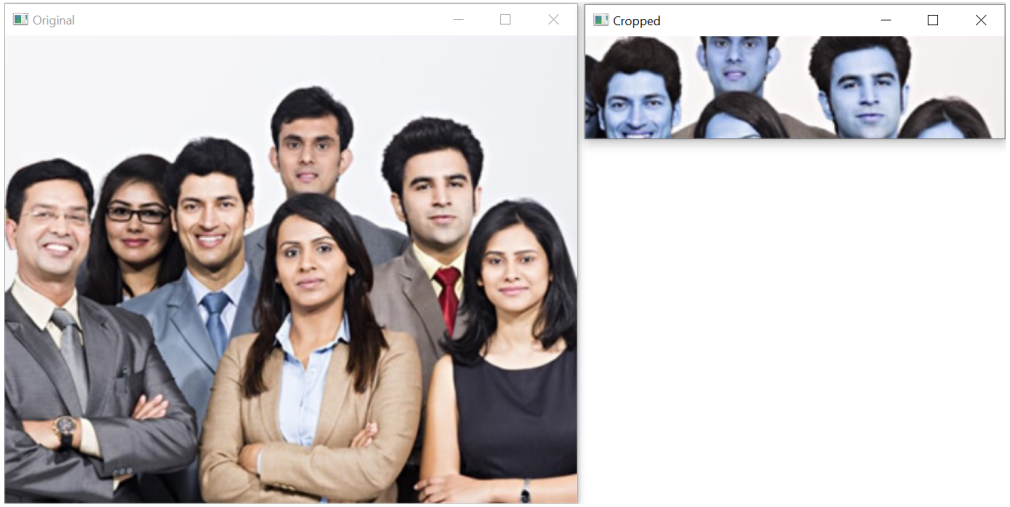
5.4 OpenCV and NumPy Operations: **Crop using indexing**

- Using NumPy and OpenCV together, we convert images from OpenCV format to NumPy arrays,using Indexing we perform our desired operations like Cropping, and then convert the arrays back to OpenCV format.

In [ ]:

```
import cv2
import numpy as np

# Load image using OpenCV
img = cv2.imread('Images\\example.jpg')

# Convert image to NumPy array
img_array = np.array(img)

# Slice the image using NumPy array indexing
```

```
cropped = img_array[100:200, 150:800]

# Convert the sliced array back to OpenCV format
cropped_img = cv2.cvtColor(np.uint8(cropped), cv2.COLOR_RGB2BGR)

# Perform NumPy operations on the image
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
##blurred = cv2.GaussianBlur(gray_img, (5, 5), 0)

# Show the original image, cropped image, and blurred image
cv2.imshow('Original', img)
cv2.imshow('Cropped', cropped_img)
##cv2.imshow('Blurred', blurred)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

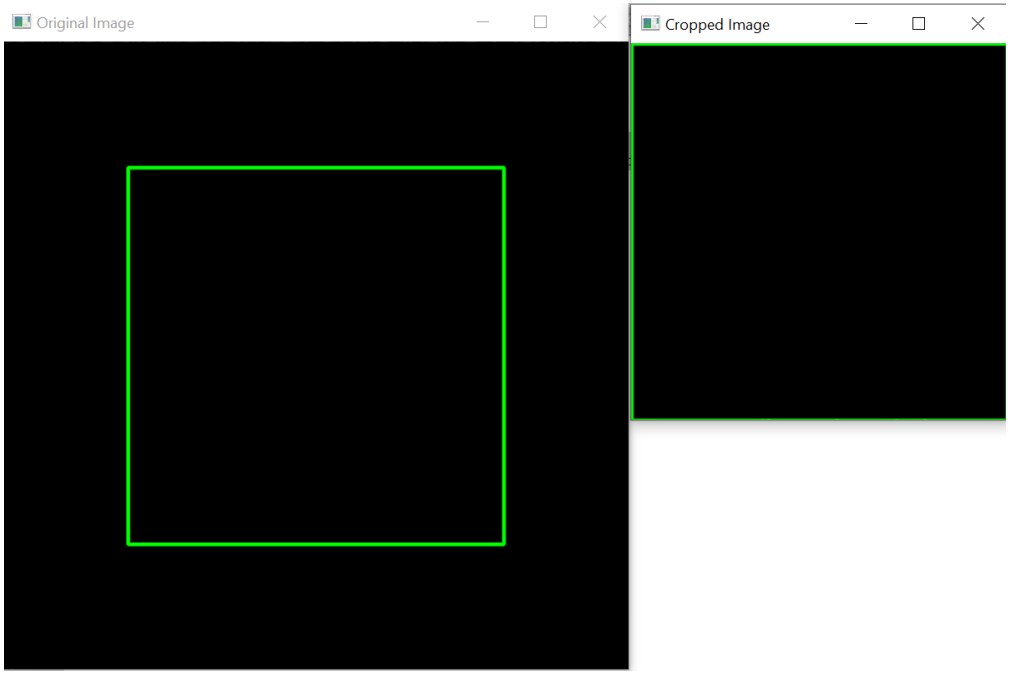


5.5 OpenCV and NumPy Operations: **Cropping Numpy Zeros**

In [ ]:

```
import cv2
import numpy as np

# Create a black image using np.zeros
img = np.zeros((500, 500, 3), dtype=np.uint8)

# Draw a rectangle on the image using OpenCV
cv2.rectangle(img, (100, 100), (400, 400), (0, 255, 0), thickness=2)
```

```
# Crop the image using NumPy array slicing
cropped_img = img[100:400, 100:400]

# Display the original and cropped images
cv2.imshow('Original Image', img)
cv2.imshow('Cropped Image', cropped_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



5.6 OpenCV and NumPy Operations: **Extracting Channel using indexing**

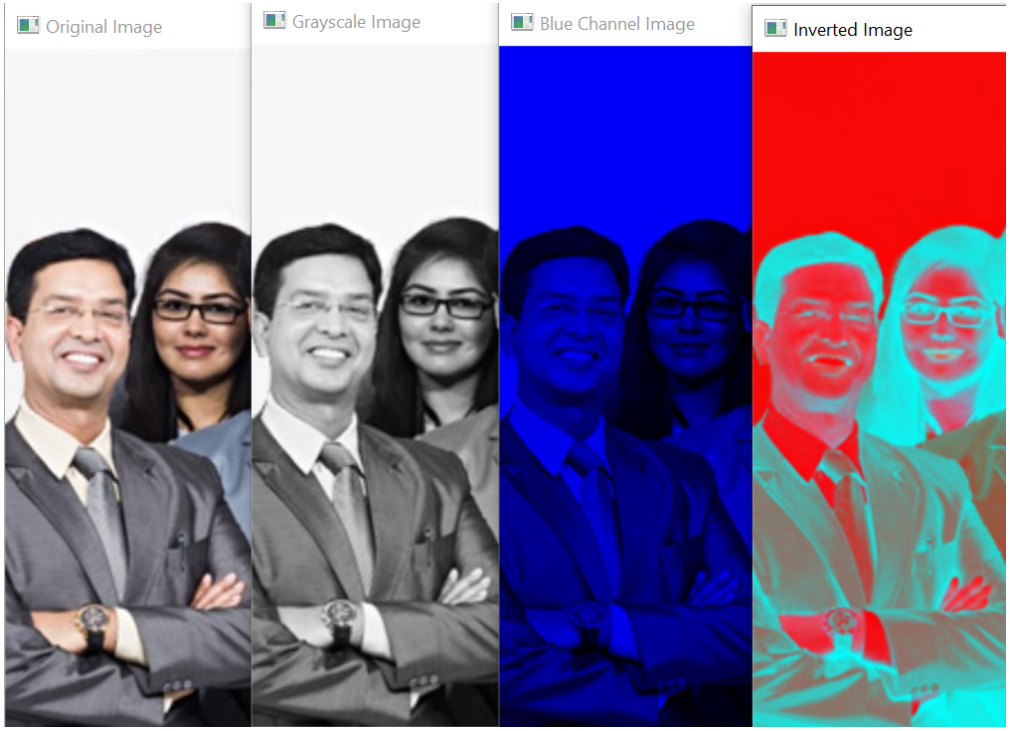
CODE EXPLANATION:

- We load an image using OpenCV's imread() function, then convert it to grayscale using the cvtColor() function. We then create two modified copies of the image using NumPy indexing: one that contains only the blue channel, and one that has the red and green channels inverted.
- To extract the blue channel, we create a copy of the image using the copy() method, then set the green and red channels to 0 using NumPy indexing: blue_img[:, :, 1] = 0 sets the green channel to 0, and blue_img[:, :, 2] = 0 sets the red channel to 0.
- To invert the red and green channels, we create another copy of the image using the copy() method, then subtract each channel from 255 using NumPy indexing: inverted_img[:, :, :2] = 255 - inverted_img[:, :, :2] subtracts each pixel value in the red and green channels from 255, effectively

inverting their colors.

In [ ]:

```
import cv2
import numpy as np

# Load an image using OpenCV
img = cv2.imread('Images\\example.jpg')

# Convert the image to grayscale
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Create a copy of the image with only the blue channel
blue_img = img.copy()
blue_img[:, :, 1] = 0 # setting the green channel zero
blue_img[:, :, 2] = 0 # setting the red channel zero

# blue_img[:, :, 0] = 0 # for setting the blue channel zero

# Create a copy of the image with the red and green channels inverted
inverted_img = img.copy()
inverted_img[:, :, :2] = 255 - inverted_img[:, :, :2]

# Display the original image and the modified images using OpenCV
cv2.imshow('Original Image', img)
cv2.imshow('Grayscale Image', gray_img)
cv2.imshow('Blue Channel Image', blue_img)
cv2.imshow('Inverted Image', inverted_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

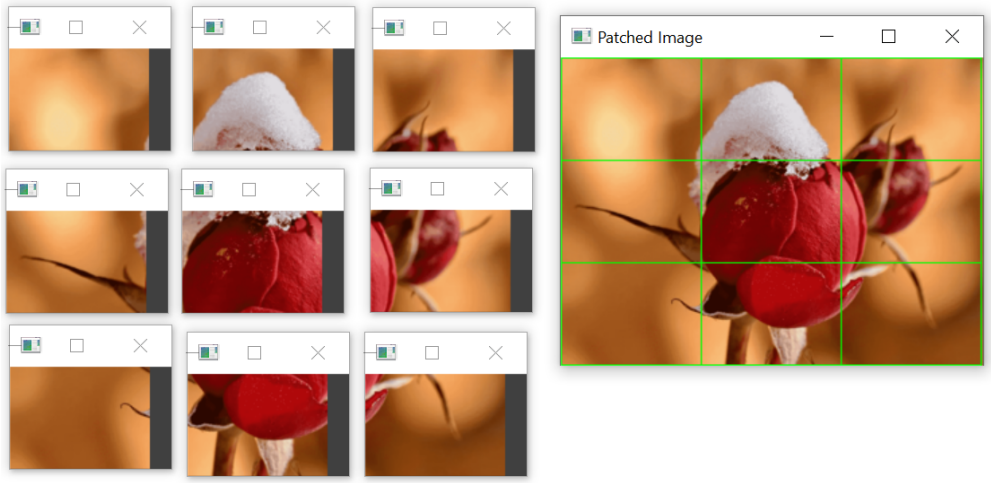


5.7 OpenCV and NumPy Operations: **Cropping & Patching**

In [ ]:

```
import cv2

img =  cv2.imread("Images\\test_image.jpg")
image_copy = img.copy()
print(img.shape)

#Set height and width to image height and width
imgheight=img.shape[0]
imgwidth=img.shape[1]

M = 76
N = 104
x1 = 0
y1 = 0

for y in range(0, imgheight, M):
    for x in range(0, imgwidth, N):
        if (imgheight - y) < M or (imgwidth - x) < N:
```

```
            break

        y1 = y + M
        x1 = x + N

        # check whether the patch width or height exceeds the image width or height
        if x1 >= imgwidth and y1 >= imgheight:
            x1 = imgwidth - 1
            y1 = imgheight - 1
            #Crop into patches of size MxN
            tiles = image_copy[y:y+M, x:x+N]
            #Save each patch into file directory
            cv2.imwrite('Images\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
            cv2.imshow('Images\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
            cv2.rectangle(img, (x, y), (x1, y1), (0, 255, 0), 1)
        elif y1 >= imgheight: # when patch height exceeds the image height
            y1 = imgheight - 1
            #Crop into patches of size MxN
            tiles = image_copy[y:y+M, x:x+N]
            #Save each patch into file directory
            cv2.imwrite('Images\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
            cv2.imshow('Images\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
            cv2.rectangle(img, (x, y), (x1, y1), (0, 255, 0), 1)
        elif x1 >= imgwidth: # when patch width exceeds the image width
            x1 = imgwidth - 1
            #Crop into patches of size MxN
            tiles = image_copy[y:y+M, x:x+N]
            #Save each patch into file directory
            cv2.imwrite('Images\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
            cv2.imshow('Images\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
            cv2.rectangle(img, (x, y), (x1, y1), (0, 255, 0), 1)
        else:
            #Crop into patches of size MxN
            tiles = image_copy[y:y+M, x:x+N]
            #Save each patch into file directory
            cv2.imwrite('Images\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
            cv2.imshow('Images\\'+'tile'+str(x)+'_'+str(y)+'.jpg', tiles)
            cv2.rectangle(img, (x, y), (x1, y1), (0, 255, 0), 1)

#Output Patched image
cv2.imshow("Patched Image",img)

cv2.waitKey()
cv2.destroyAllWindows()
```

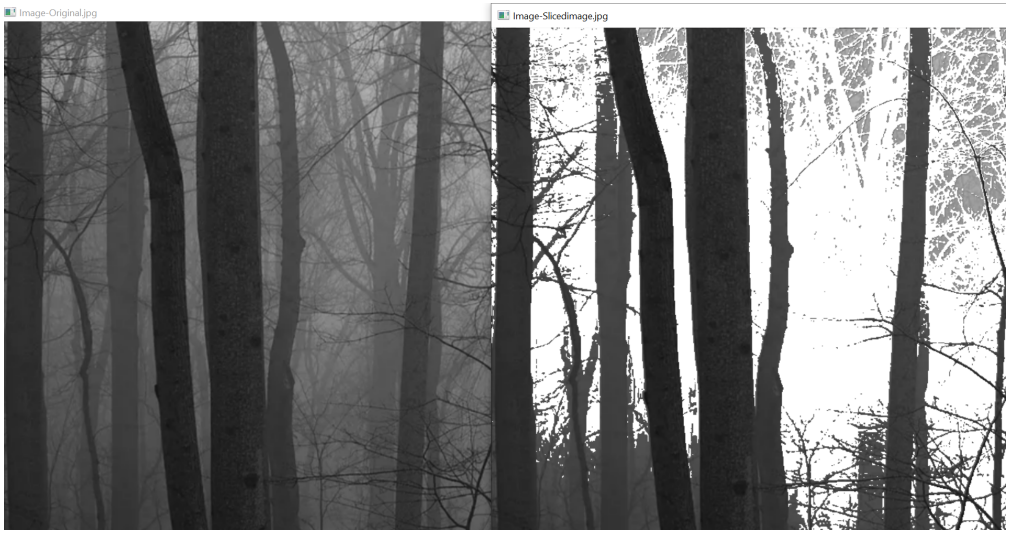


5.7 OpenCV and NumPy Operations: **Gray Slicing**

In [ ]:

```
import cv2
import numpy as np

img = cv2.imread('Images\\Image-background.png', 0)

#  Find width and height of image
row, column= img.shape

#  Create an zeros array to store the sliced image
img1 = np.zeros((row,column),dtype = 'uint8')

#  Specify the min and max range
min_range = 80
max_range = 140

# Loop over the input image and if pixel value lies in desired range set it to 255
# otherwise set it to desired value
for i in range(row):
    for j in range(column):
        if img[i,j]>min_range and img[i,j]<max_range:
            img1[i,j] = 255
        else:
            img1[i,j] = img[i-1,j-1]
```

```
cv2.imshow('Image-Original.jpg', img)
cv2.imshow('Image-Slicedimage.jpg', img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



5.7 OpenCV and NumPy Operations: **Tile Slicing** - CHECK IF REMOVE

In [ ]:

```
import cv2
import numpy as np

# Load the image
img = cv2.imread('Images\\clahe.jpg')
cv2.imshow('Original Image', img)
# Define the size of each tile
tile_size = (256, 256)

# Slice the image into tiles
tiles = []
for i in range(0, img.shape[0], tile_size[0]):
    for j in range(0, img.shape[1], tile_size[1]):
        tile = img[i:i+tile_size[0], j:j+tile_size[1]]
        tiles.append(tile)

# Convert the list of tiles to a NumPy array
tiles_array = np.array(tiles)
```

```
# Display the first tile

cv2.imshow('Tile 0', tiles_array[0])
cv2.imshow('Tile 1', tiles_array[1])

cv2.waitKey(0)
cv2.destroyAllWindows()
```

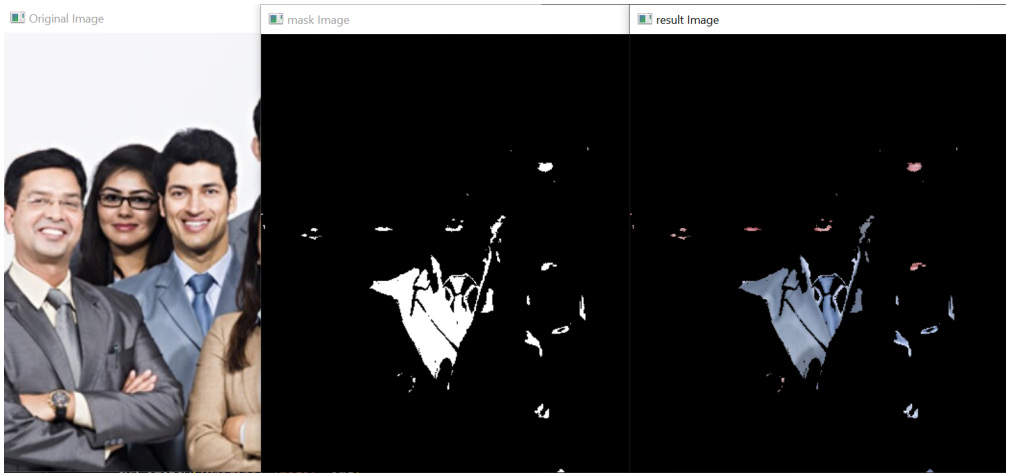
```
<ipython-input-8-61d5564b9bd8>:18: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or
-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  tiles_array = np.array(tiles)
```

5.8 OpenCV and NumPy Operations: **Masking using Filtering**

In [ ]:

```
import cv2
import numpy as np

img = cv2.imread('Images\\example.jpg')

##while(1):
##    _, frame = cap.read()
# It converts the BGR color space of image to HSV color space
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Threshold of blue in HSV space
lower_blue = np.array([60, 35, 140])
upper_blue = np.array([180, 255, 255])

# preparing the mask to overlay
mask = cv2.inRange(hsv, lower_blue, upper_blue)

# The black region in the mask has the value of 0,
# so when multiplied with original image removes all non-blue regions
result = cv2.bitwise_and(img, img, mask = mask)

cv2.imshow('Original Image', img)
cv2.imshow('mask Image', mask)
cv2.imshow('result Image', result)

cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```



Homework Problems

1) Make a view,change the view and display both arrays

```
arr = [21, 23, 34, 74, 5]
Update the zeroth index to 65
```

2) Make a copy, change the original array, and display both arrays

```
arr = [78, 27, 23, 74, 75]
Update the zeroth index to 29
```

3) Write Code to perform Sorting on the below 2 D array

```
arr = [[39, 20, 74], [75, 90, 81]]
```

4) Sort the below array in alphabetical order

```
arr = ['figs', 'cherry', 'mango']
```

5) Write code for Creating Filter Directly from Array, so will return only even elements from the original array a=[22,32,54,25,33]

6) Create a NumPy Filter Array for the below numPy Array using the below Boolean Filter pattern

```
arr = np.array([4, 3, 40, 6]) x=[True,False,False,True]
```

7) Create code to Flatten the below 2D array to 1-D Format

```
a=([[1,4,7], [2,5,8],[3,6,9]])
```

For solutions of Homework questions, please refer to the `HomeworkSolution.ipynb` file