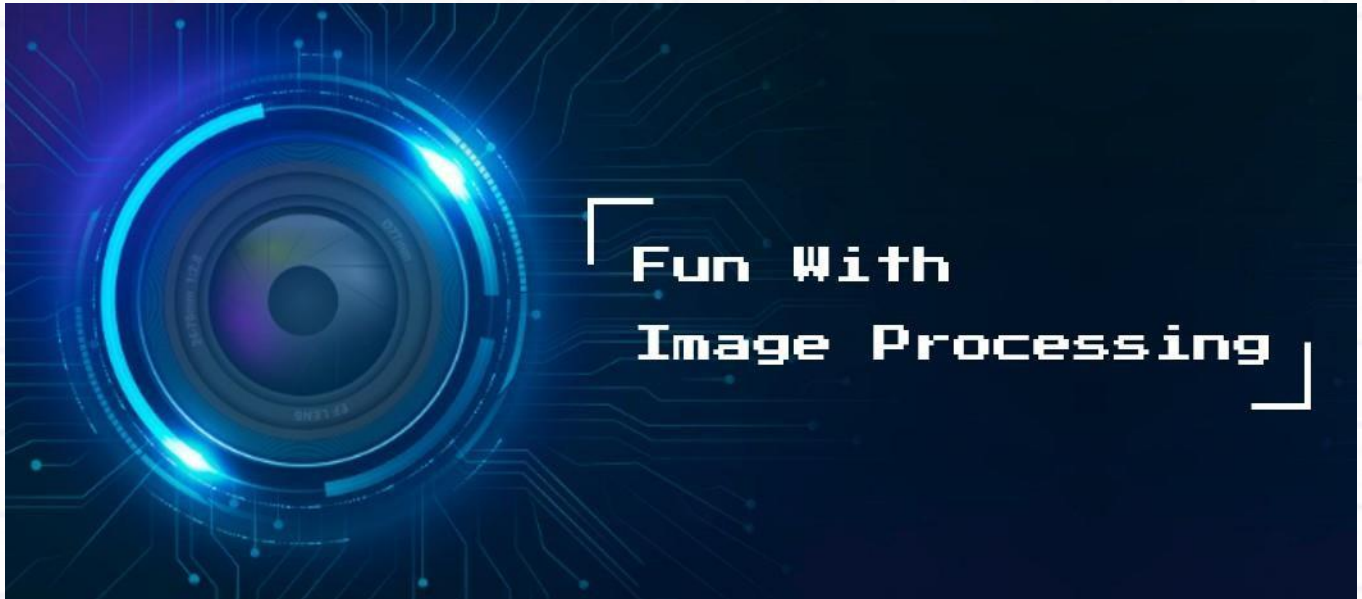


# Fun With Image Processing

---

**Requirements: A Linux (Ubuntu) (Preferred) or Windows Machine with a webcam.**

**Download the image and video files used for the sample codes from the course repository here:**  
[Fun\\_With\\_Image\\_Processing](#)



---

Introduction to the course	This is an introductory course designed for people from any background who is completely new to the area of image processing. You will be learning various functions of OpenCV to manipulate images and videos.
What does this course aim to achieve?	It aims to give hands-on practice of various OpenCV tools and functionalities with many example problems and their solutions to work on and understand the use of tools and functions of OpenCV. It also aims to make the learning experience fun by introducing many interesting warmup exercises at the end of each section.
What is being built in this course?	There are three final capstone build Projects: (i) A program to detect and count pulse on a pre-recorded video; (ii) A program to cartoonize an image; (iii) And finally, a program that detects and tracks a ball bouncing on the ground.
Course Prerequisites	The course requires a student to know the basics of python, PIP (package manager for Python), basic shell commands in Linux, and version control with GIT.

---

# Contents

---

## Pre-requisite.

1. Shell
2. Python-Pip
3. Git

## Core Computer Vision Learning Track:

1. [Introduction](#)
2. [Reading - High-level overview of Video Encoding and ffmpeg](#)
  - Video Encoding - Compression and Resolutions
  - Making Video Intuitive: An Explainer
3. [Setting up Python, OpenCV, and Visual Studio Code](#)
  - [Installing Python](#)
    - Windows
    - Linux (Ubuntu)
  - [Installing OpenCV](#)
  - [Installing and setting up Visual Studio Code](#)
    - Windows
    - Linux (Ubuntu)
4. [Reading and saving image files using OpenCV](#)
  - [Reading image files](#)
  - [Changing colorspace](#)
  - [Saving/writing image files](#)
  - [Warmup Exercise 1](#)
5. [Reading and saving the video feed](#)

- [Launching the camera](#)
  - [Getting video properties](#)
  - [Changing video properties](#)
  - [Accessing and manipulating pixels](#)
  - [Saving the video](#)
  - [Warmup Exercise 2](#)
6. [Drawing Functions](#)
- [Line, Arrowed Line, Polylines, Rectangle, Circle, Put Text](#)
  - [Warmup Exercise 3](#)
7. [Interacting with the video](#)
- [Mouse Events](#)
  - [Draw a rectangle with a mouse](#)
  - [Draw a curve using a mouse](#)
  - [Interacting with trackbars in OpenCV](#)
  - [Warmup Exercise 4](#)
8. [Object detection and tracking with color mask](#)
- [Understanding HSV Colorspace](#)
  - [Masking color](#)
  - [Finding contours and bounding box](#)
  - [Warmup Exercise 5](#)
9. [Frame manipulation and transformation](#)
- [Resize](#)
  - [Rotate](#)
  - [Edge detection](#)
  - [Finding and drawing contours](#)

- [Smoothing](#)
- [Perspective warping](#)
- [Warmup Exercise 6](#)

#### 10. [Face detection and recognition](#)

- [Face Detection](#)
- [Face Recognition](#)

### **[Capstone Build Projects](#)**

#### I. [Posterization \(Cartoonization\) of portraits](#)

- Problem statement
- Guided steps

#### II. [Finding Heart Rate \(Pulse\)](#)

- Problem statement
- Guided steps

#### III. [Bounce Count](#)

- Problem statement
- Guided steps

### **[Uploading projects to GitHub](#)**

### **[What Next?](#)**

### **[End of Build Series](#)**

# Pre-requisite:

---

**The course requires a student to know the basics of python, PIP (package manager for Python), basic shell commands in Linux, and version control with GIT.**

- You will be using Python programming language for this build task along with OpenCV-Python for manipulating and transforming images (frames) in a video.

## Why Python?

Python is one of the most popular programming languages because of its simplicity which makes it very easy to learn. It is more like plain English and hence easy to read as well. One doesn't actually need to have any programming knowledge and still be able to easily pick up python. It can achieve tasks in fewer lines of code as compared to other languages. It also supports an extensive number of libraries to work with, thanks to its huge and ever-growing community base.

- You will also use PIP which is a package manager for Python-based packages. It is like playstore for Python, using which you can install/ uninstall libraries for python. This will be useful to manage packages like OpenCV and NumPy, which you will be using throughout this build series.

## Why OpenCV?

OpenCV is a very popular cross-platform open-source computer vision (CV) and machine learning (ML) library used for real-time vision applications. It has more than 2500 optimized CV and ML algorithms. These algorithms can be used to perform tasks like; detecting and recognizing faces, tracking and identifying objects, tracking camera movements, classifying human actions, extracting 3d models of objects, etc. It also has a huge user community. It has C++, Python, Java, and MATLAB interfaces and supports Windows, Linux, Android, and Mac.

- You should also know basic shell commands to create, modify and change permission associated with files and folders which might come in handy if you are using a Linux Operating System for this course (Which is also recommended), but you can also finish this course on a windows machine.

## Why Linux?

Linux is an operating system just like windows, but it is open source and more secure. You can easily find a Linux machine around you. For eg; computers, servers, tablets, and smartphones (Android), smart TVs, smartwatches and fitness trackers, cameras, embedded devices, robots, gaming consoles, amazon kindle, self-driving cars, navigation systems, routers and modems, IoT devices, and even supercomputers. For a programmer, Linux supports almost all the major programming languages, and the image processing task that you may perform in the future will likely be on one of these machines. Hence, it is good to get familiar with Linux Operating System.

- Finally, at the end of the series, you will finish capstone tasks (Final Projects) on your own by following the instructions provided and uploading them to your personal GitHub profile to showcase your work. For which a basic knowledge of version control with GIT will be useful.

### **More about GIT:**

Git is a version control system that tracks changes in a file as it gets updated. It is commonly used for collaboratively developing the source code of the software. Git makes it easy to work on the same project with multiple people at the same time and independent of each other's version. Multiple collaborators can have their own branch of development, and hence cannot interfere with other collaborators' work and finally, all of them can be merged together.

**Hence it is assumed that the person following this builder series is familiar with the Pre-requisites. If not, the following resources might be useful to get started:**

#### **1. SHELL:**

- Install Linux Bash Shell on Windows 10: <https://itsfoss.com/install-bash-on-windows/>
- Course overview + the shell: <https://missing.csail.mit.edu/2020/course-shell/> TLCL
- Book-Chapter 1 (For further understanding): <https://linuxcommand.org/tlcl.php>

#### **2. PYTHON-PIP:**

- Python: <https://github.com/iitmvcg/Python-Exercises>
- Pip: [https://www.w3schools.com/python/python\\_pip.asp](https://www.w3schools.com/python/python_pip.asp)
- NumPy (For further understanding): [https://numpy.org/devdocs/user/absolute\\_beginners.html](https://numpy.org/devdocs/user/absolute_beginners.html)
- SciPy (For further understanding): <https://www.mygreatlearning.com/blog/scipy-tutorial/>
- SciPy API (For further understanding): <https://docs.scipy.org/doc/scipy/reference/index.html>

#### **3. GIT:**

- Quick Tutorial: <https://www.w3schools.com/git/>
- For a quick demo: <https://www.youtube.com/watch?v=c6b6B9oN4Vg>
- For interactive git learning (Optional): <https://learngitbranching.js.org/>

# Core Computer Vision Learning Track:

---

## 1. Introduction:

- This is an introductory course is designed for people who are from any background and are completely new to the area of image processing.
- The Overall Course is divided into 3 parts.
  - Core Computer Vision Learning Track
  - Capstone Build Task
  - Uploading Projects to GitHub
- Each section will walk you through some useful tools and functions and then example programs using those tools to achieve simple tasks. The Example codes are well commented to give you as much information as possible.
- At the end of each section, there are warmup exercises for which the solutions are not provided. These exercises are designed in such a way that it tests your knowledge and understanding of the concepts until that point of the course.
- We will be using OpenCV, a popular and open-sourced image processing library along with python programming language. And for writing the code we will be using Visual Studio Code which is an IDE that contains everything in one place and also some helper plugins that will assist and make your coding work easy.
- At the end of the course, you will be equipped with all the necessary tools to perform the Build Tasks on your own. And finally, we will guide you to upload your projects/work repository to your personal GitHub account.
- Hope you enjoy this!

## 2. Reading - High-level overview of Video Encoding and ffmpeg:

**Objective:** In this, you will learn what is a video and a few important concepts around it such as resolution, compression, frame rate (FPS), Bitrate, etc.

- **Video Encoding - Compression and Resolutions:**

<https://eyevinntechnology.medium.com/chessboard-for-beginners-video-encoding-compression-andresolutions-bcefe04fa639>

- **Making Video Intuitive: An Explainer:**

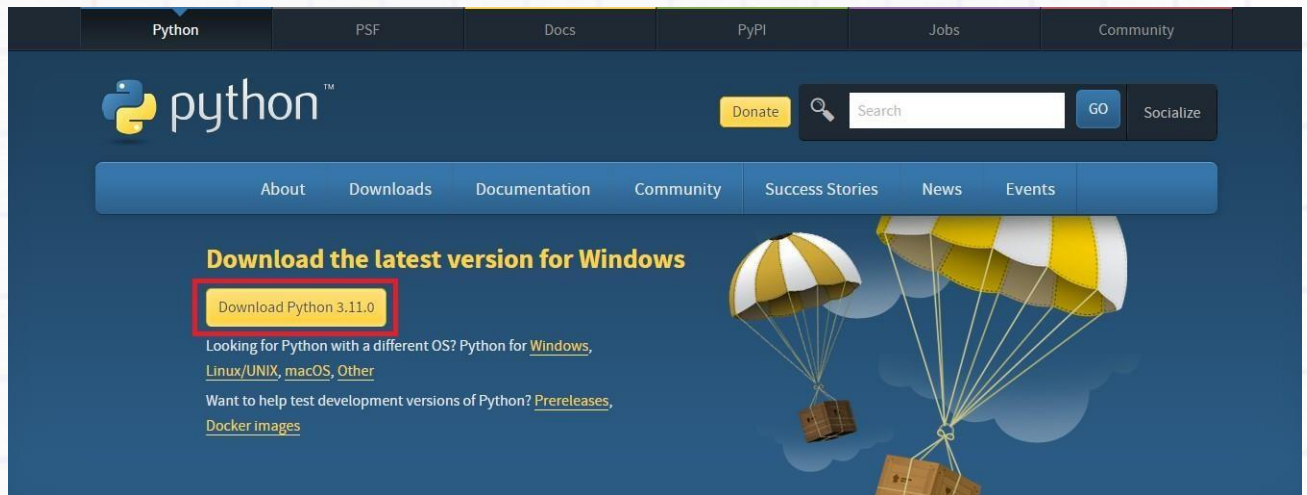
<https://blog.cloudflare.com/making-video-intuitive-an-explainer/>

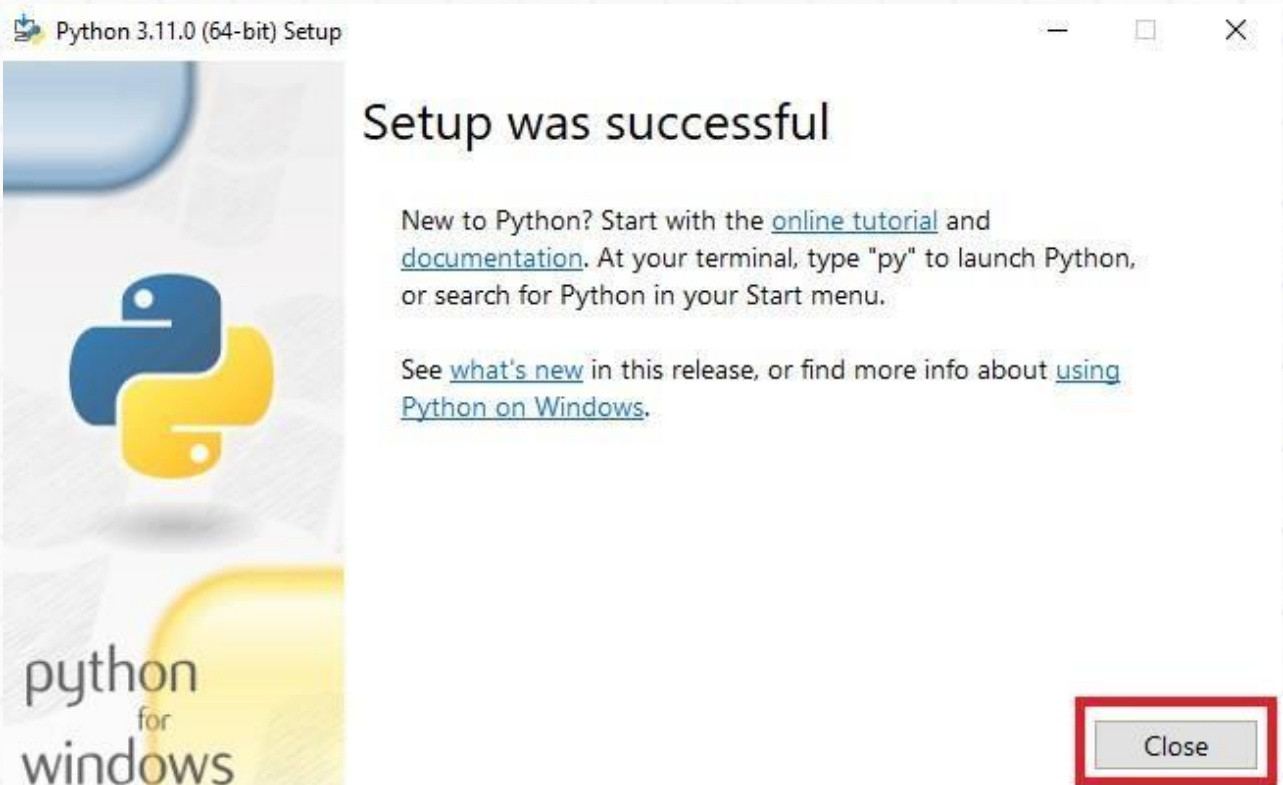
### 3. Setting up Python, OpenCV, and Visual Studio Code:

#### Installing Python:

##### Windows:

- Download and install Python from: <https://www.python.org/downloads/>





### Linux (Ubuntu):

- Linux distributions are likely to come with preinstalled Python. To check, open the terminal and run `python` or `python3` if it is installed already, you would see something similar to this:

```
**user**::~~$ python3
Python 3.11.0a7 (main, Apr 20 2022, 17:44:14)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- **If not**, follow the series of commands given below:

```
**user**::~~$ sudo apt update
**user**::~~$ sudo apt install python3
```

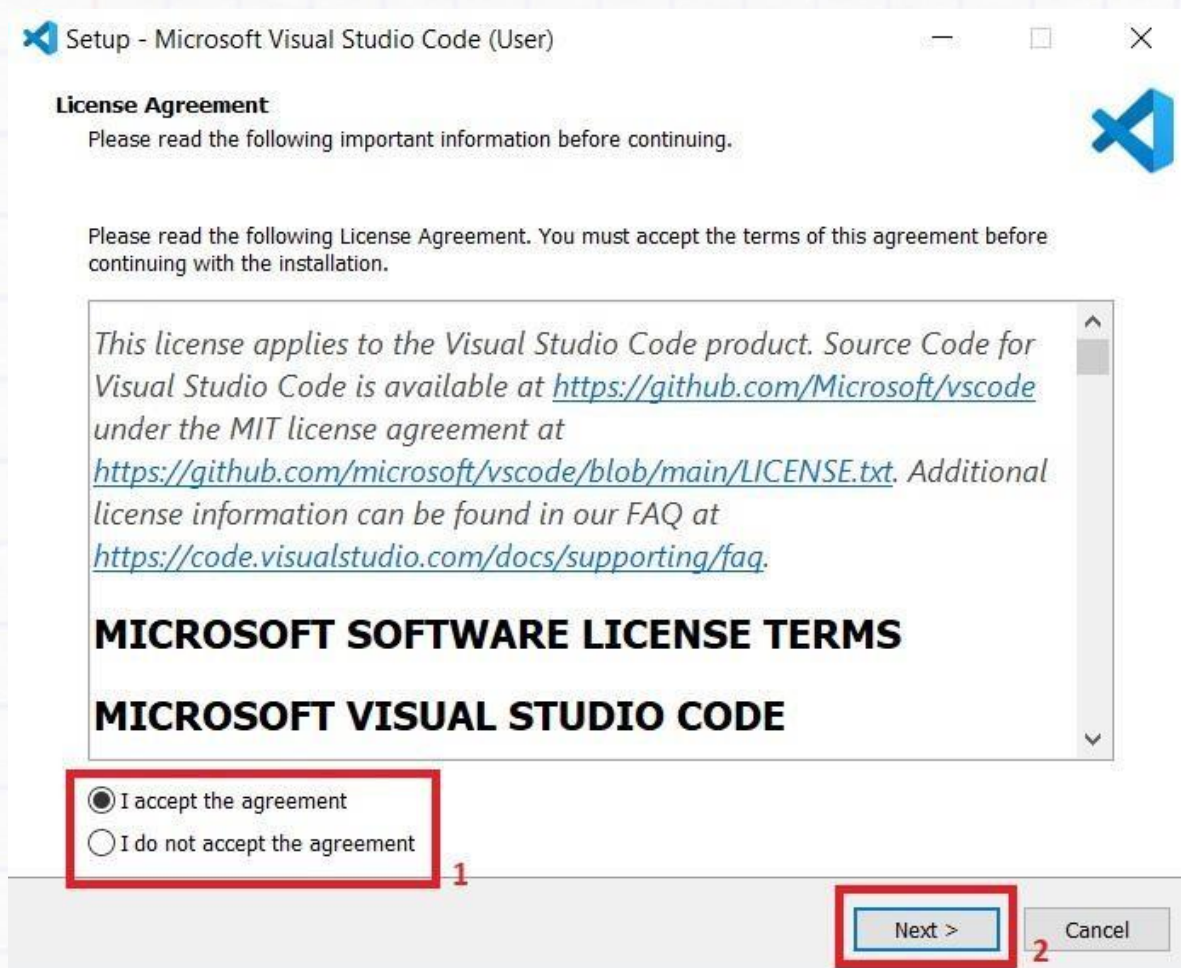
## Installing OpenCV:

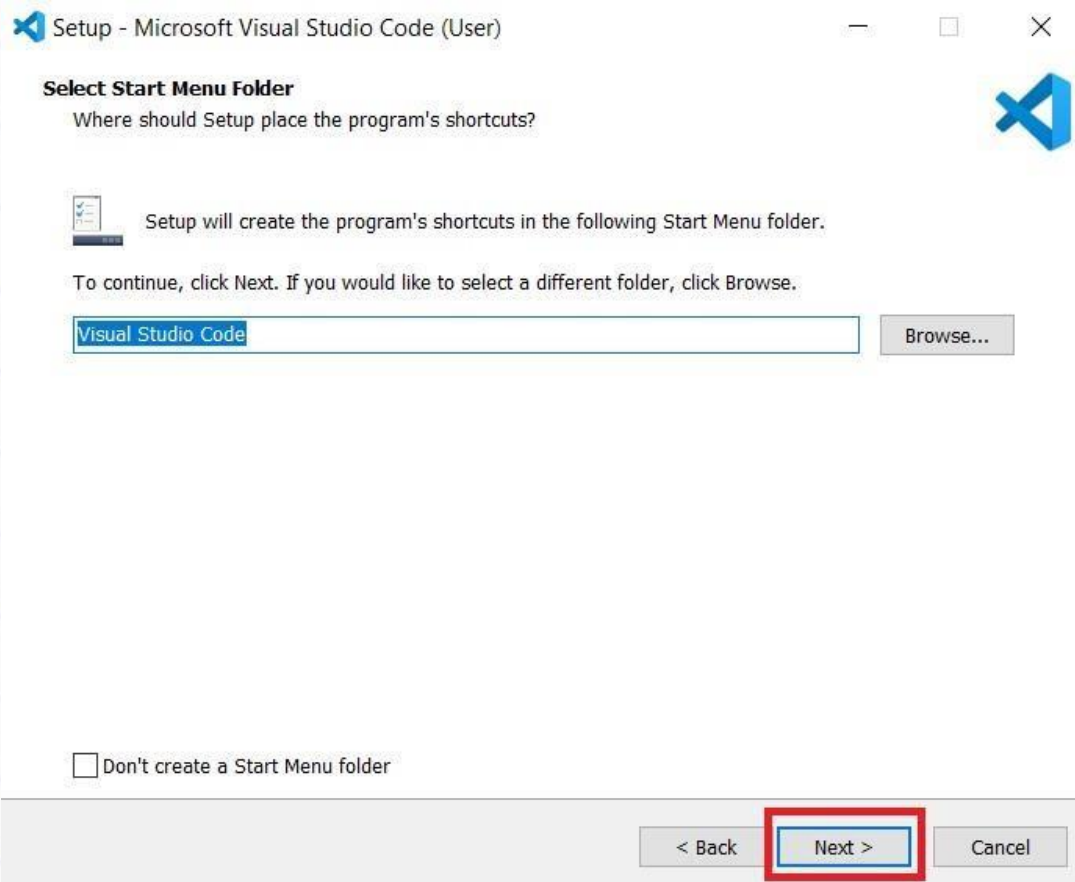
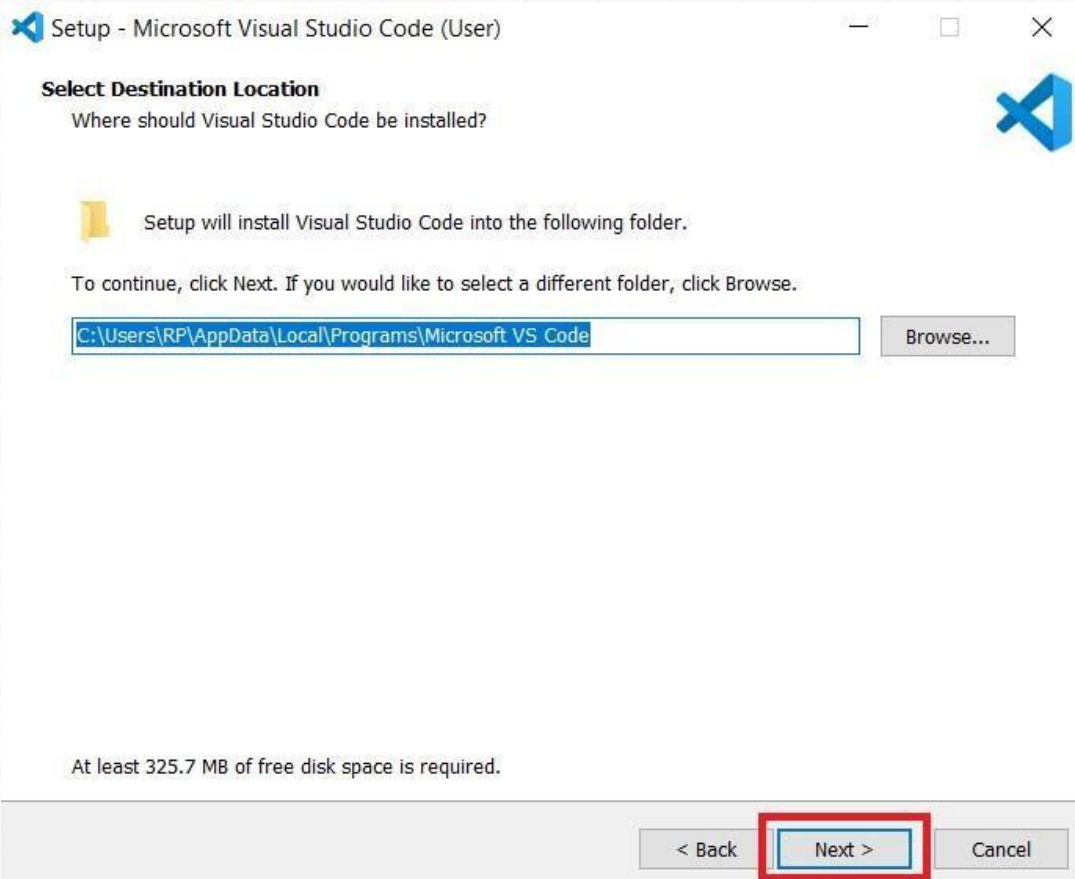
- Open Terminal or Command Prompt and run `pip install opencv-python` to install OpenCV Library.
- For more details or troubleshooting refer: <https://pypi.org/project/opencv-python/>

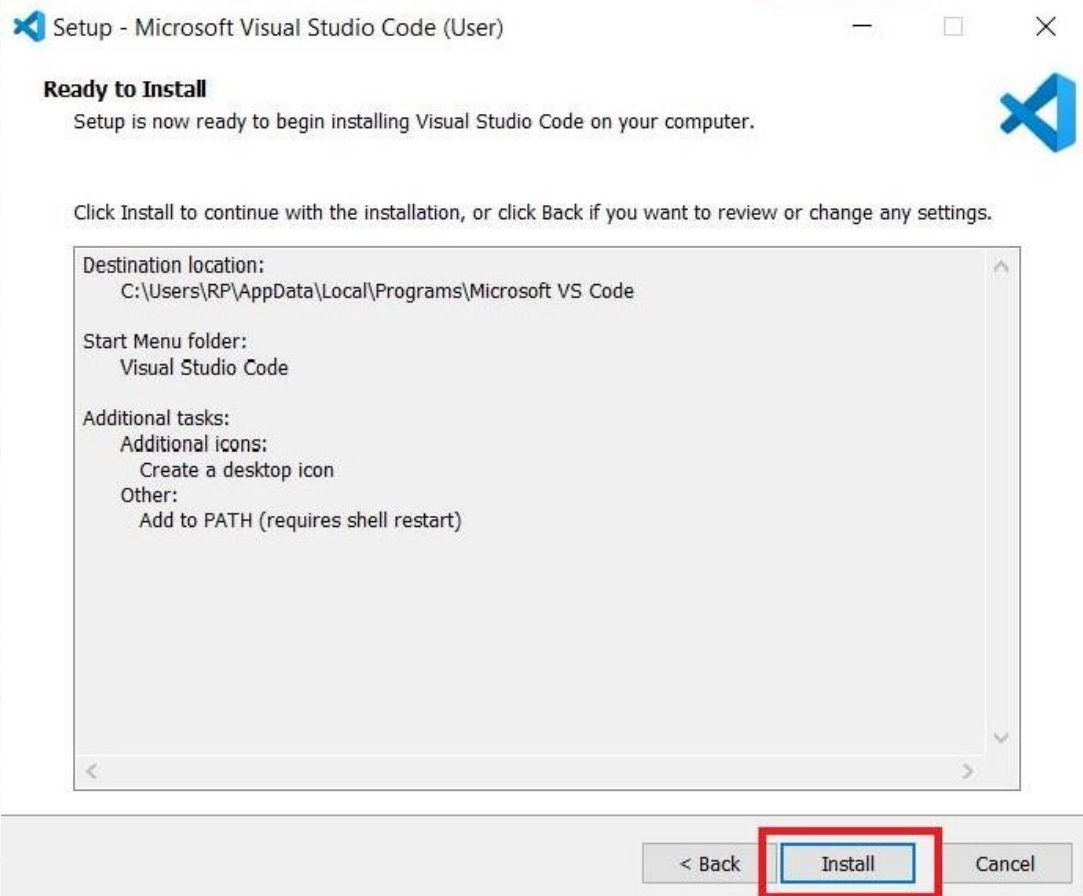
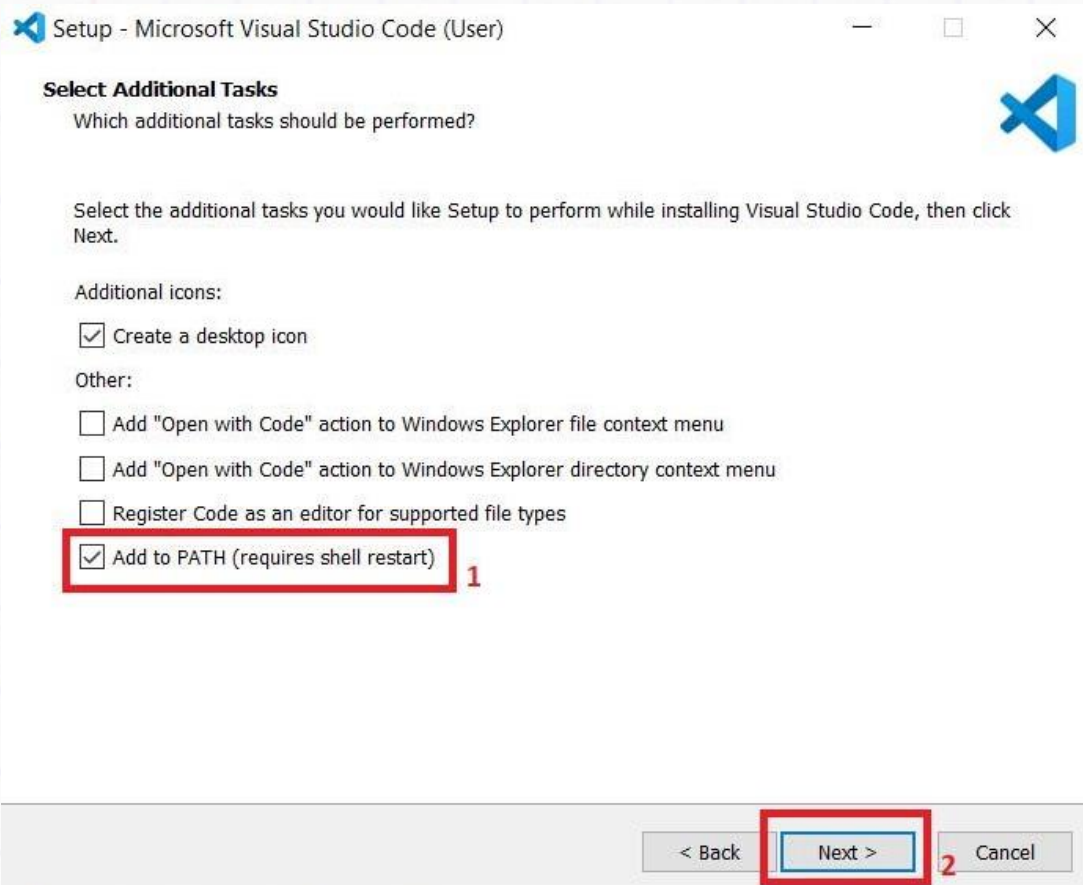
## Installing and setting up Visual Studio Code:

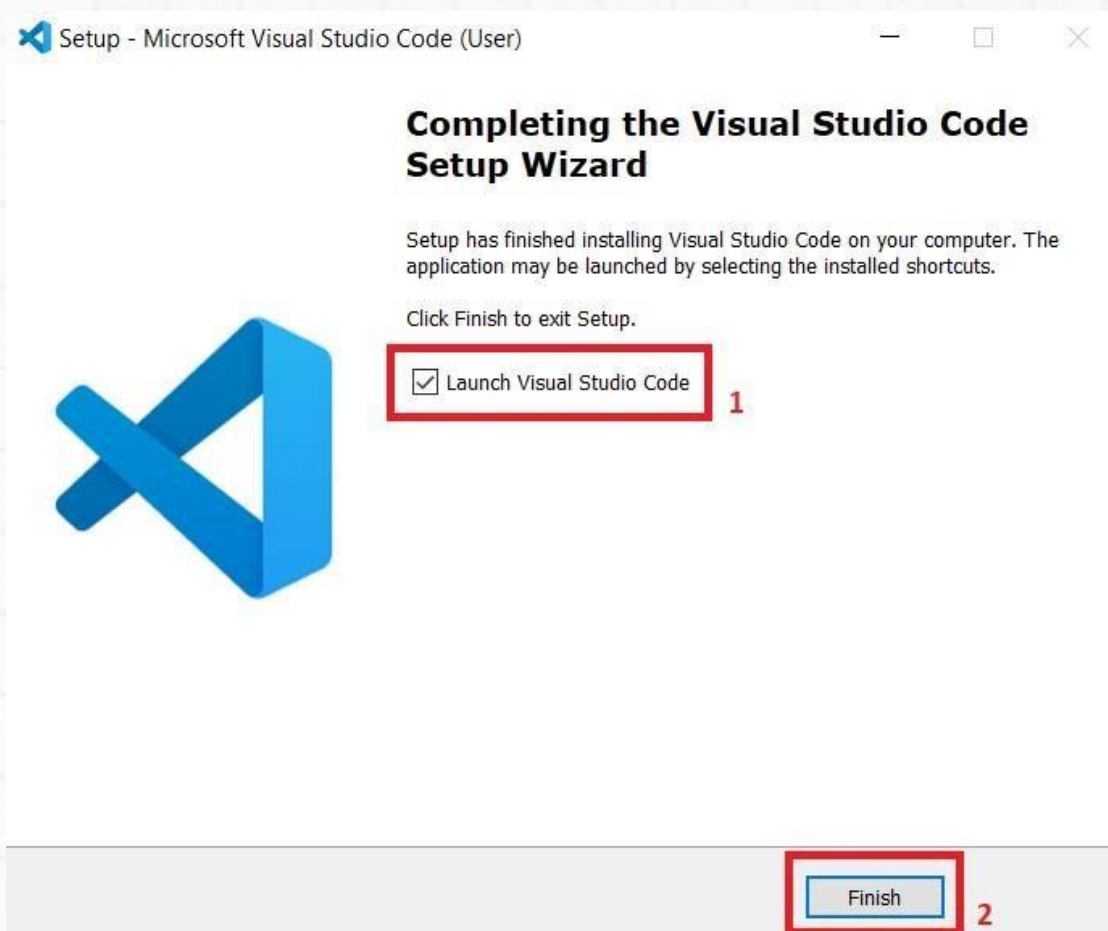
### Windows:

- Download and install VS-Code from: <https://code.visualstudio.com/download>

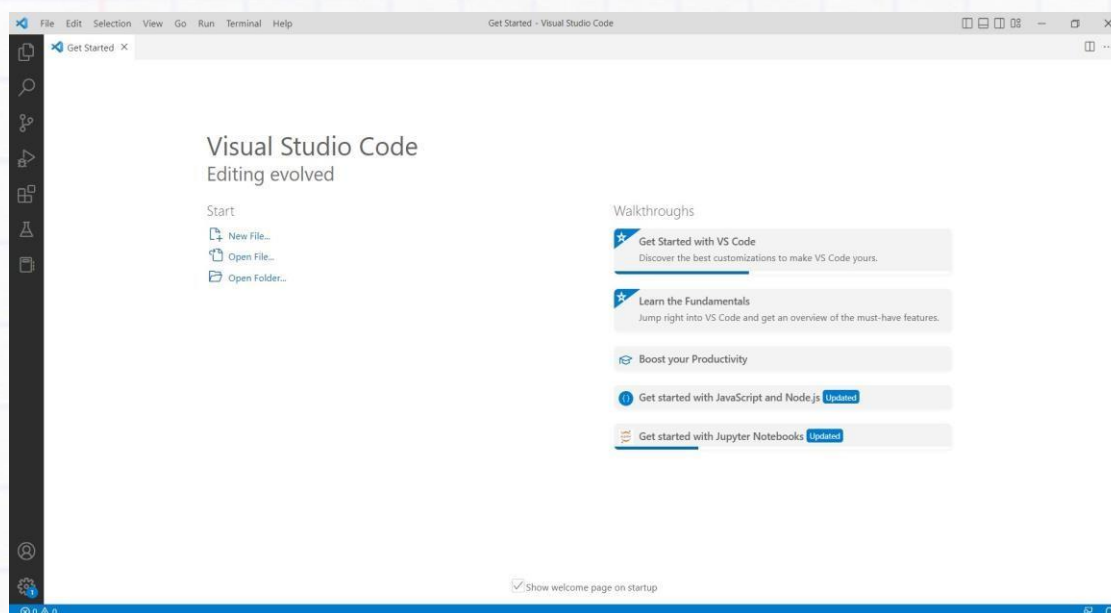




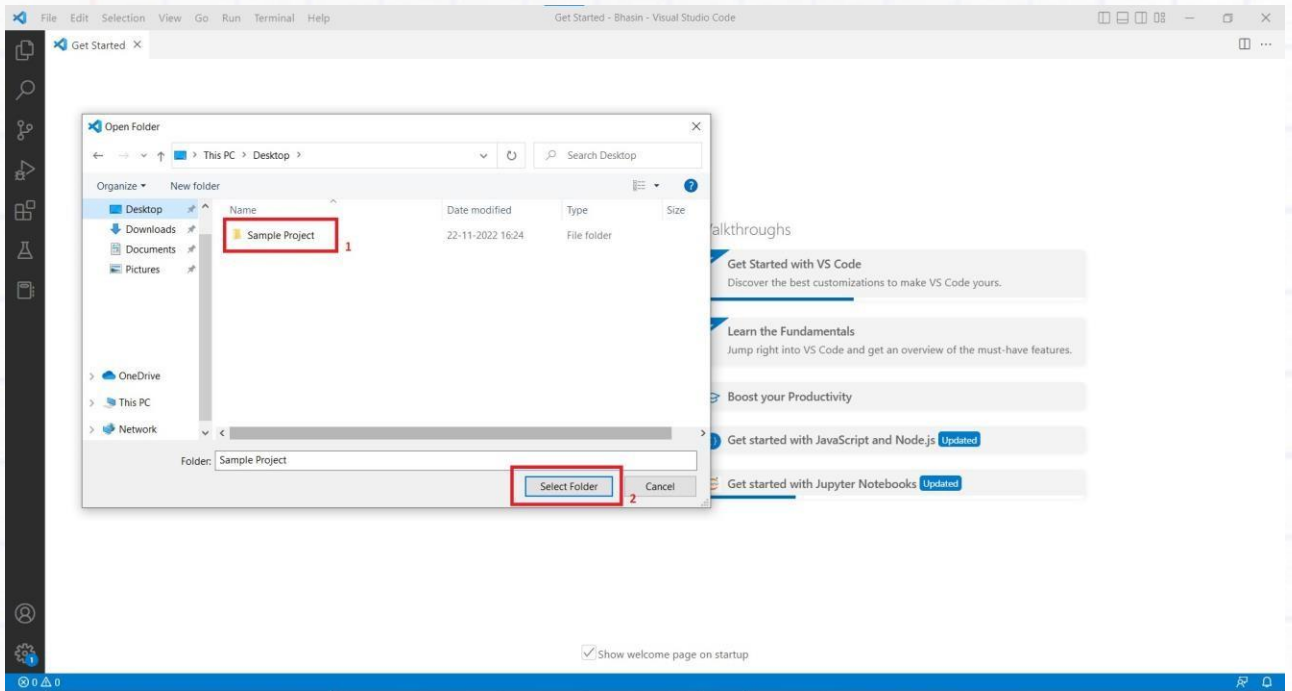
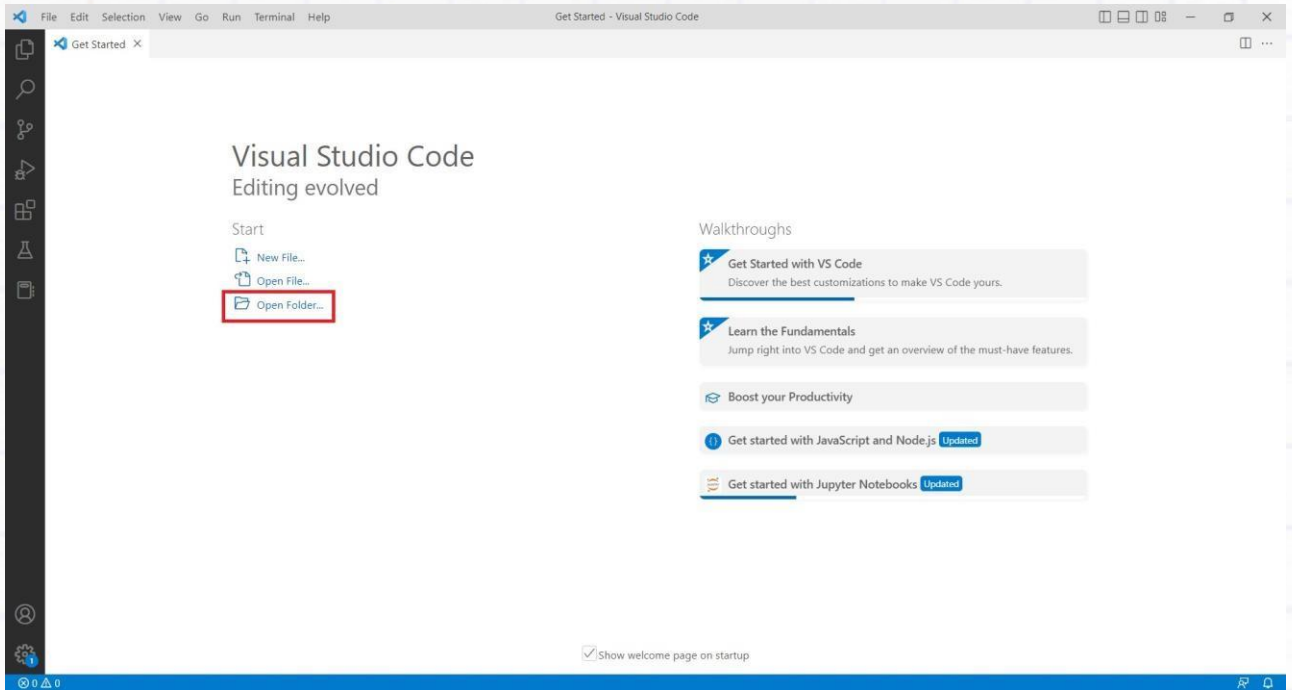


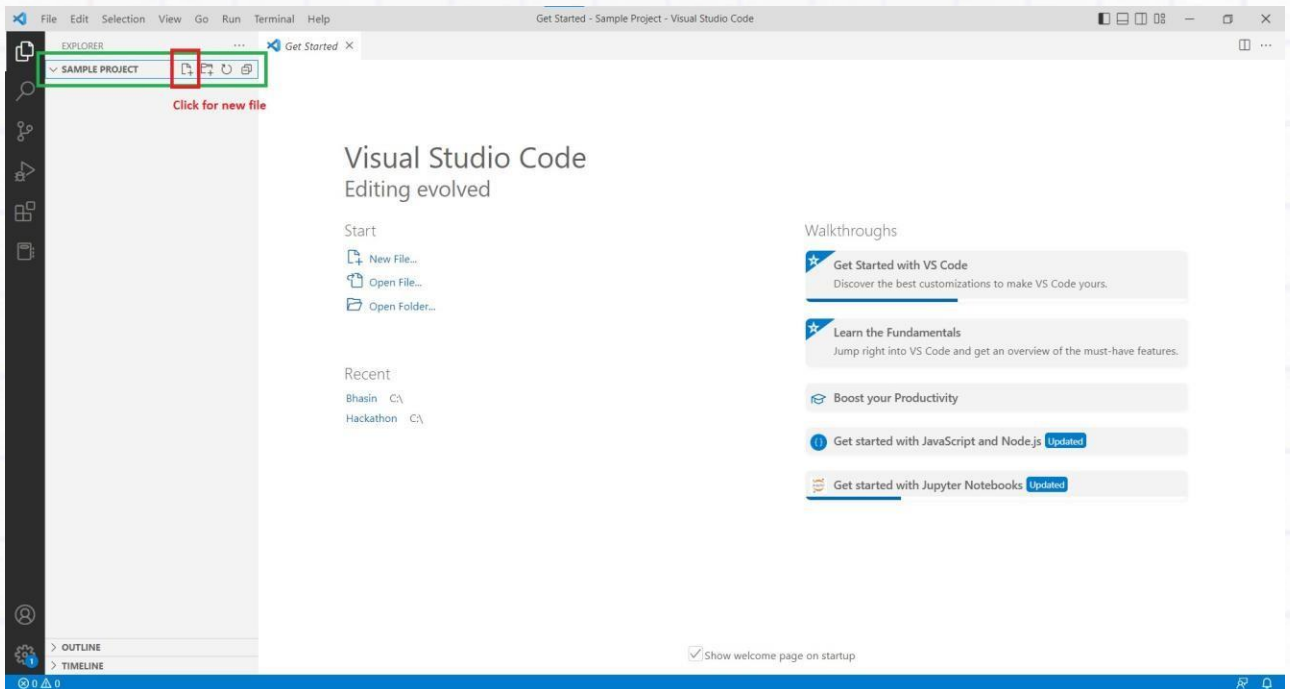


- After Installation, the welcome page similar to this opens up:

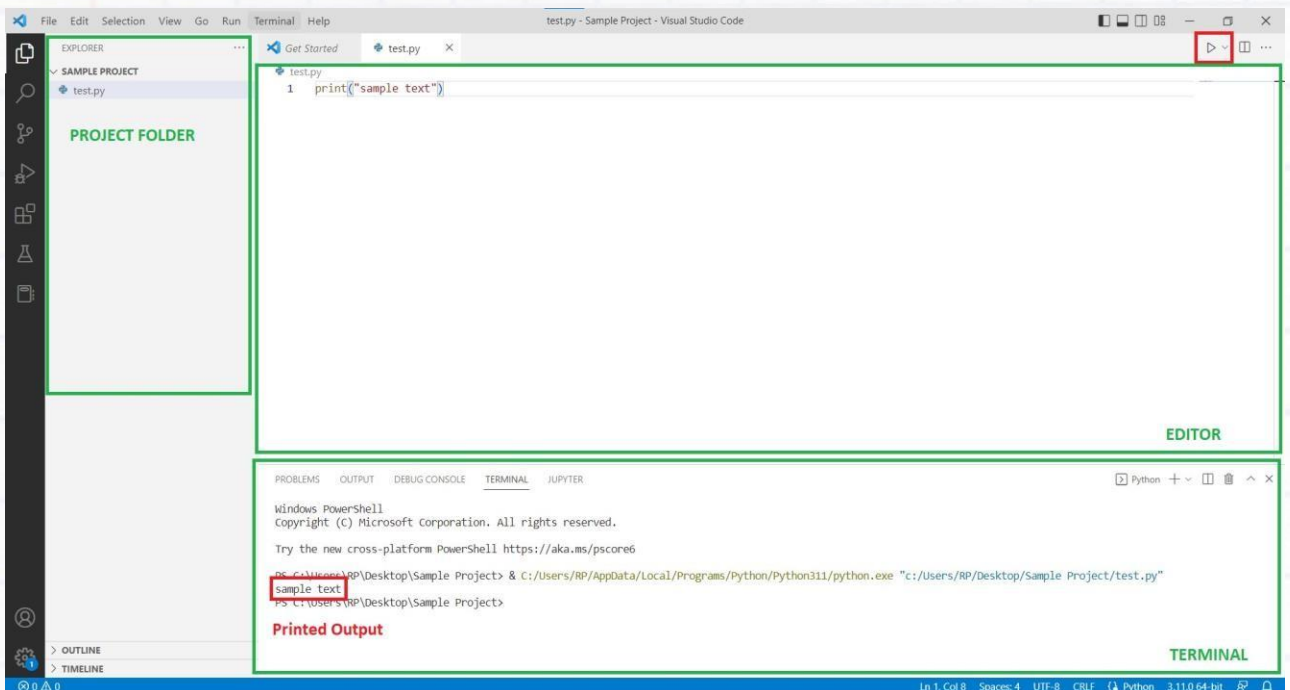


- Open your project folder and create a new file as "test.py". The extension ".py" after the filename is important as it helps in identifying the programming language used within the file.

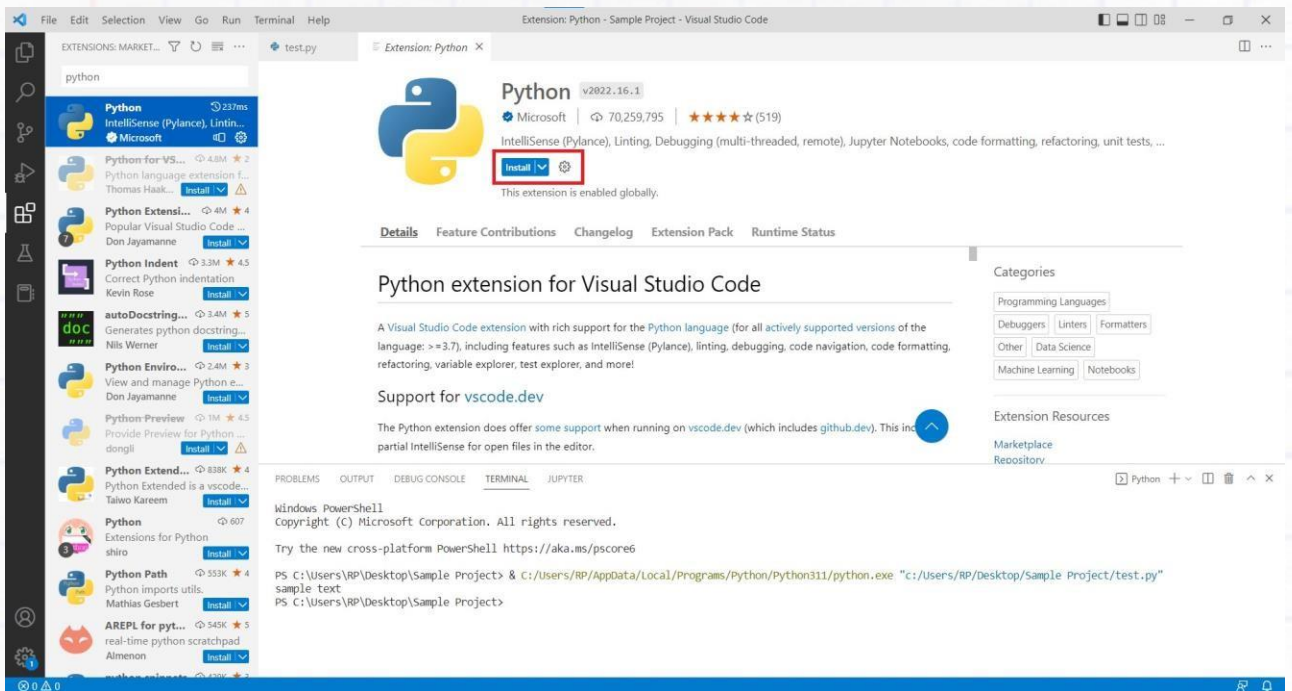
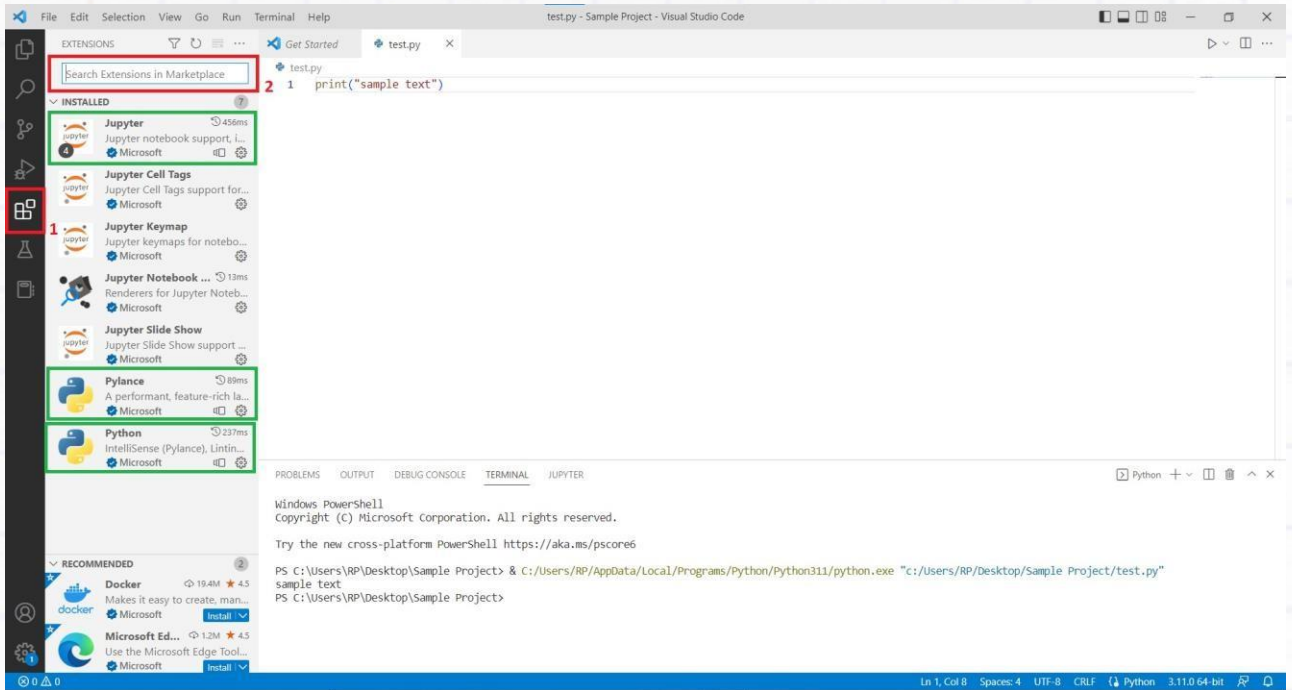




- After creating "test.py", type in `print("sample text")` in it and save using `ctrl+S`. And then click on the triangular button on the top right corner of the screen to run the "test.py" file



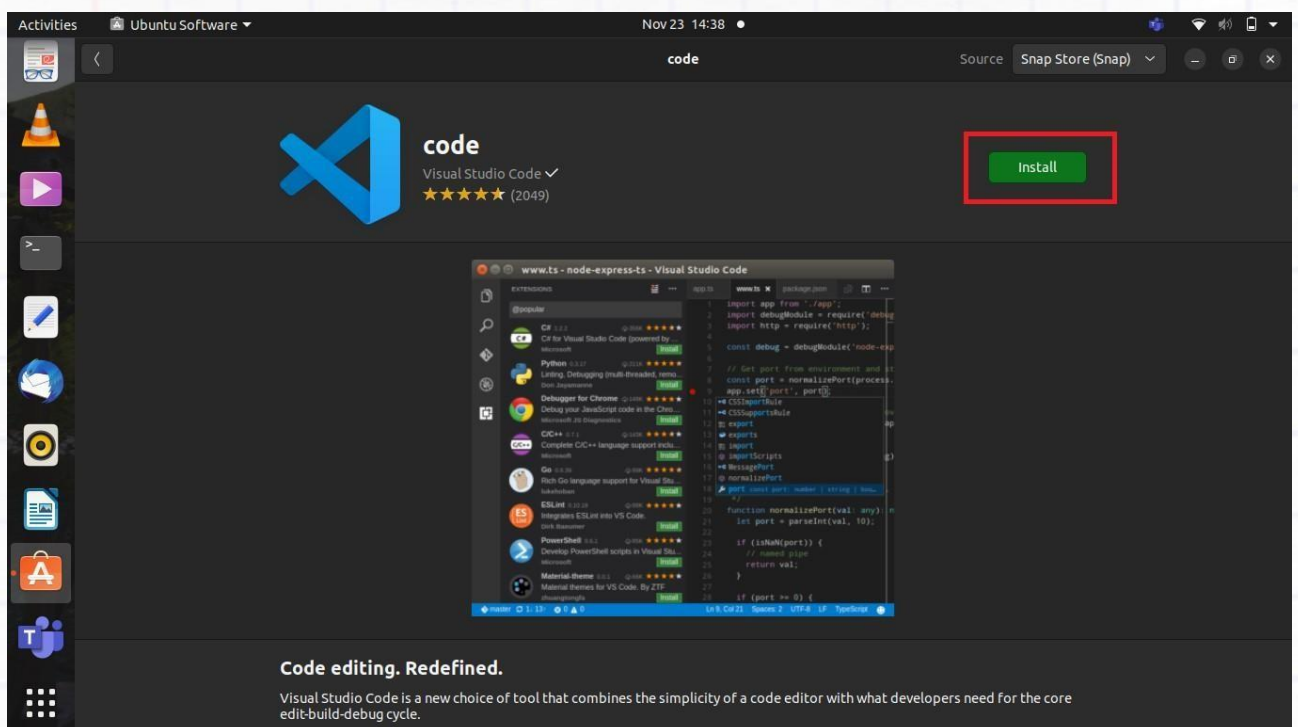
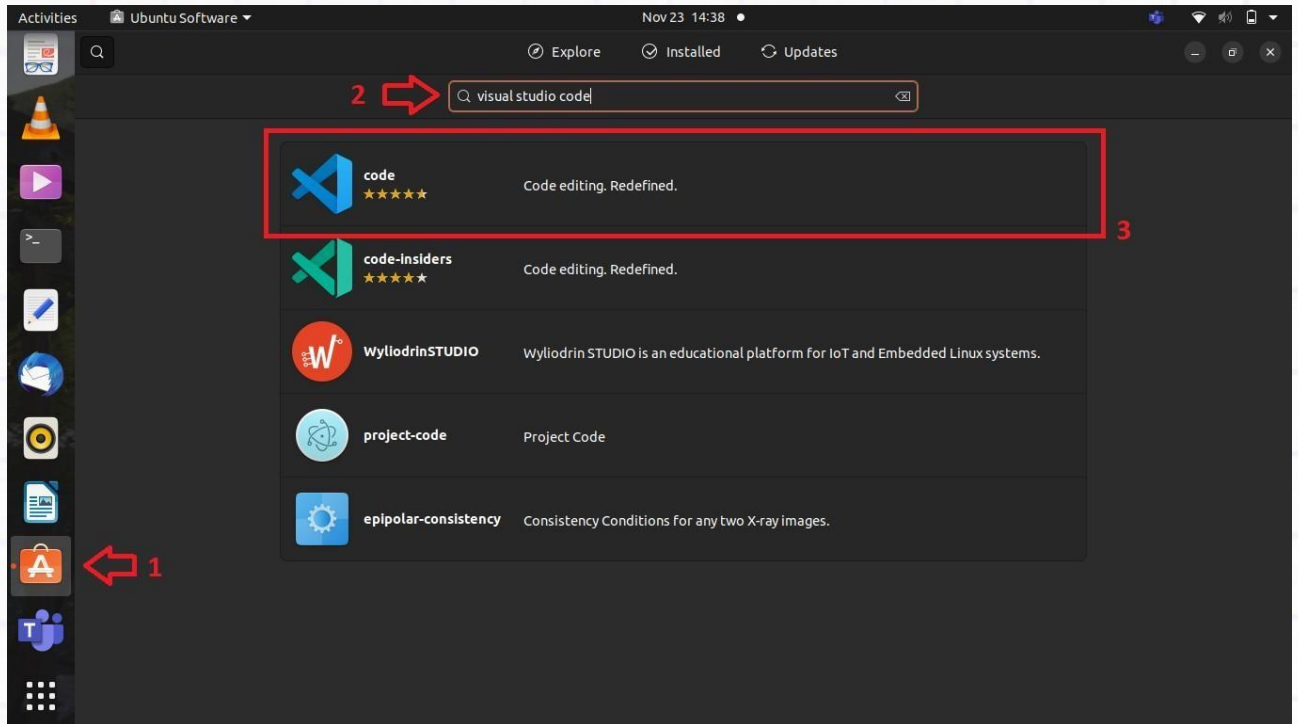
- Now install a few useful extensions like "Python", "Pylance", and "Jupyter" for VS Code. Click on the extensions tab > search for the extensions in the search tab > Open it and click on install.



- Now we are ready for the tasks.

## Linux (Ubuntu):

- Open Ubuntu Software and search for Visual Studio Code, and install it from there.



- After installation, the remaining instructions are the same as instructions for Windows.

## 4. Reading and saving image files using OpenCV:

**Objective:** In this section, you will learn how to read an image and save it after converting it to Grayscale. Check out the example code below.

### READING IMAGE FILES

- Before starting, create a folder on the desktop named "CV\_Builder\_Series", and another folder named "task\_1" in it.
- Open VS Code > press **Ctrl + K + O** to open a project folder > Navigate to the desktop and open "CV\_Builder\_Series".
- Create a new file named `task_1.py` inside the folder "task\_1".
- First import `cv2` library, and then you will use `imread` function to read an input image, `imshow` function to display the image on a window, `waitKey` function to hold the program until there's any key press, and finally `destroyAllWindows` to kill all the active windows at the end of the program.

#### syntax:

```
cv2.imread(path, flag)
```

for more details on `flag`, refer: [Flags used for image file reading and writing](#)

```
cv2.imshow(window_name, image_name)
```

#### example code:

```
# Importing OpenCV Library
import cv2

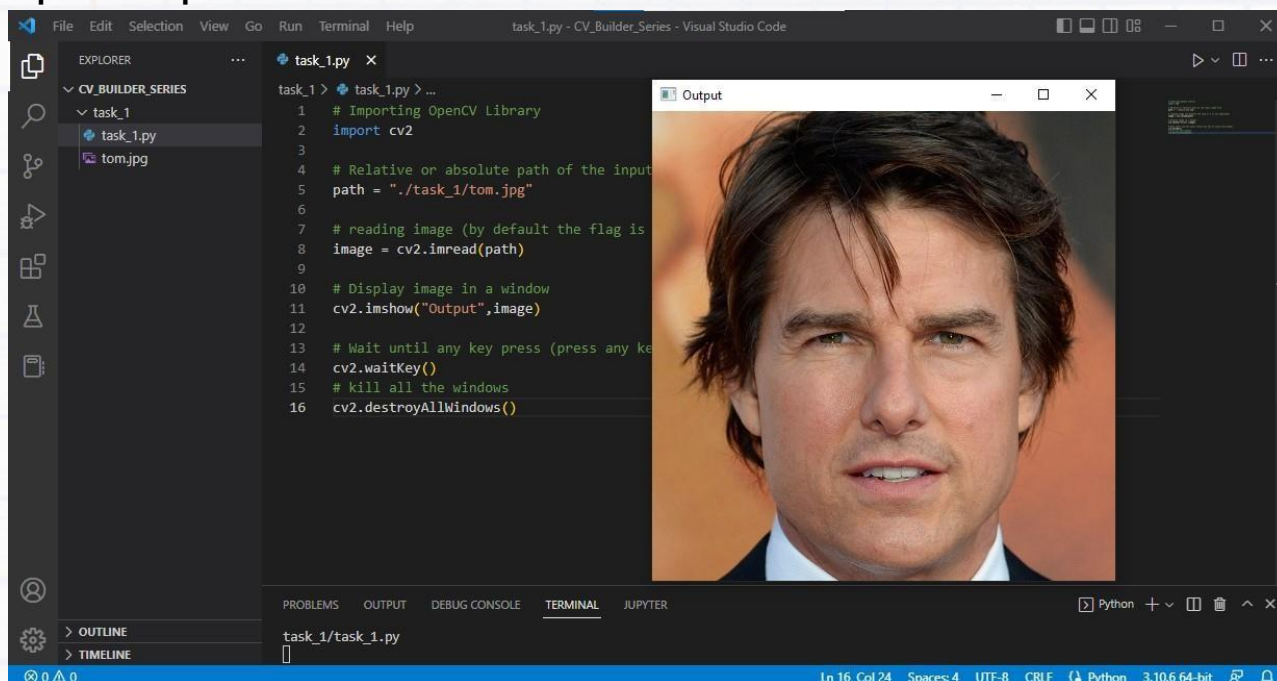
# Relative or absolute path of the input image file
path = "./task_1/tom.jpg"

# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)

# Display image in a window
cv2.imshow("Output", image)

# Wait until any key press (press any key to close the window)
cv2.waitKey()
# kill all the windows
cv2.destroyAllWindows()
```

## Expected Output:



## CHANGING COLORSPACE

- Now try using `cvtColor` to convert the image to Grayscale using `COLOR_BGR2GRAY` option in the previous code and display both images using `imshow`

### syntax:

```
cv2.cvtColor(source_image,color_conversion_code)
```

For more information on `color_conversion_code`, refer: [Color Space Conversions](#)

### example:

```
# Importing OpenCV Library
import cv2

# Relative or absolute path of the input image file
path = "./task_1/tom.jpg"

# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)

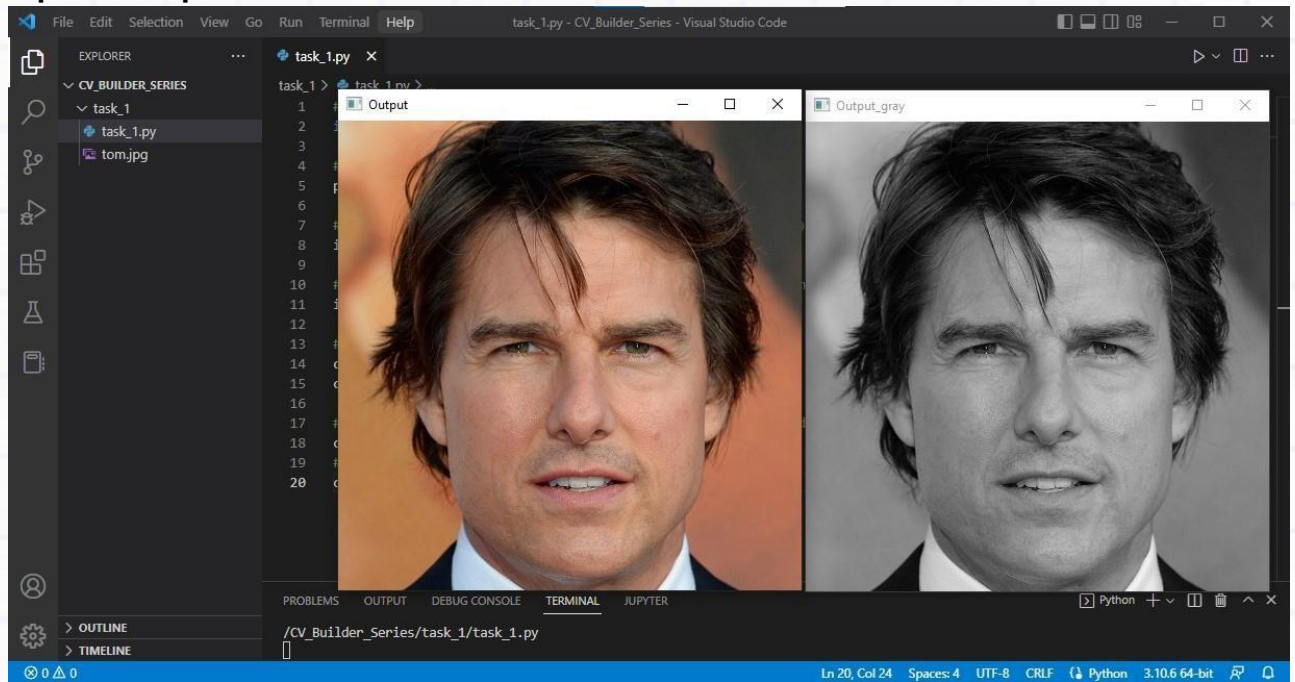
# converting image to Grayscale (also OpenCV reads image in BGR format and
hence BGR to Gray)
image_gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# Display image in a window
cv2.imshow("Output",image)
cv2.imshow("Output_gray",image_gray)

# Wait until any key press (press any key to close the window)
cv2.waitKey()

# kill all the windows
cv2.destroyAllWindows()
```

## Expected Output:



## SAVING/WRITING IMAGE FILE

- Lets save the gray image using `imwrite` function.

### **syntax:**

```
cv2.imwrite(path + file_name, image_name)
```

### **example: task\_1.py**

```
# Importing OpenCV Library
import cv2

# Relative or absolute path of the input image file
path = "./task_1/tom.jpg"

# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)

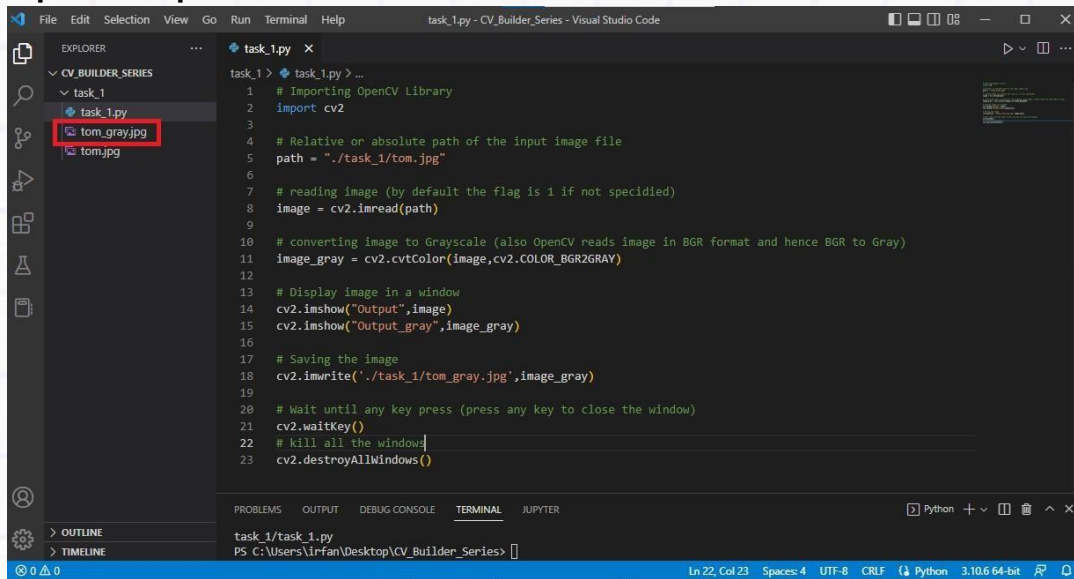
# converting image to Grayscale (also OpenCV reads image in BGR format and
hence BGR to Gray)
image_gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# Display image in a window
cv2.imshow("Output",image)
cv2.imshow("Output_gray",image_gray)

# Saving the image
cv2.imwrite('./task_1/tom_gray.jpg',image_gray)

# Wait until any key press (press any key to close the window)
cv2.waitKey()
# kill all the windows
cv2.destroyAllWindows()
```

## expected output:

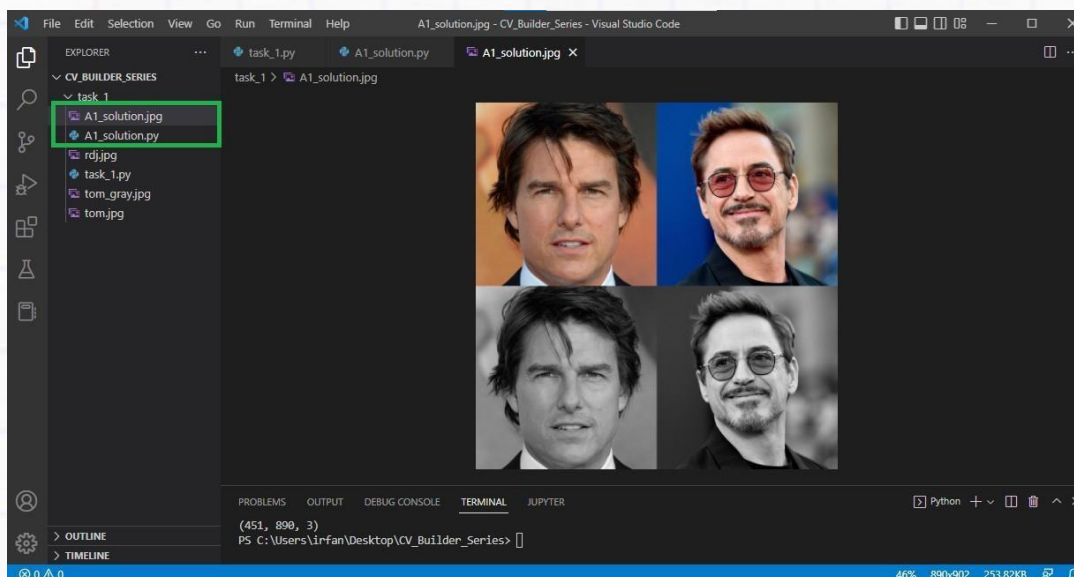


```
task_1.py
1 # Importing OpenCV Library
2 import cv2
3
4 # Relative or absolute path of the input image file
5 path = "./task_1/tom.jpg"
6
7 # reading image (by default the flag is 1 if not specified)
8 image = cv2.imread(path)
9
10 # converting image to Grayscale (also OpenCV reads image in BGR format and hence BGR to Gray)
11 image_gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
12
13 # Display image in a window
14 cv2.imshow("Output",image)
15 cv2.imshow("Output_gray",image_gray)
16
17 # Saving the image
18 cv2.imwrite('./task_1/tom_gray.jpg',image_gray)
19
20 # Wait until any key press (press any key to close the window)
21 cv2.waitKey()
22 # kill all the windows
23 cv2.destroyAllWindows()
```

## WARMUP\_EXERCISE\_1:

- Read 2 different images.
- Create a new image by arranging them side-by-side.
- Convert that new image to Gray.
- Finally, arrange the colored pair of images on top and the pair of gray images on the bottom and save it as "A1\_solution.jpg". **Tip:** Remember to match the shape of the Grayscale image to that of the colored image while concatenating.

## Expected Output:



## 5. Reading and saving the video feed:

**Objective:** In this section, you will learn how to get a video feed from a camera as well as from a video file, some useful functions to get and also set the video properties, and access and manipulate the pixel values.

### LAUNCHING THE CAMERA

- For this task, you will be using the function `VideoCapture` to create a video capture object that connects the camera. It can either take in a camera as a parameter (an integer) or a video file path, or any IP camera.
- Another function `read` is used to read frames one by one from the video capture object on which we can perform image processing. A function `imshow` is used to display the frame in a window.
- We also need something to break out of the loop where the `waitKey` function comes in handy. It takes in time in milliseconds as a parameter and waits for the given time before closing the window. This function also returns the ASCII value of the key pressed on the keyboard, which can be used to detect a particular key press to trigger some action.
- Finally, `cv2.destroyAllWindows()` is used to close all active OpenCV windows.

#### Syntax:

```
cam =cv2.VideoCapture(input)
```

- `input` can be a camera number or a video file or an IP camera.

```
cam.read()
```

- This function returns two arguments: True or False which indicates a successful frame read, and a numpy array of the current frame.

```
cv2.imshow('window_name', frame_name)
```

- `window_name` > Name of the window in which you want to display the current frame.
- `frame_name` > Frame that you want to display in the current loop.
- Checkout the following example code that reads the camera and displays the feed in a window:

### Example Code:

```
# import required libraries here
import cv2

# video capture object where 0 is the camera number for a usb camera (or
webcam)
# if 0 doesn't work, you might need to change the camera number to get the
right camera you want to access
cam = cv2.VideoCapture(0)

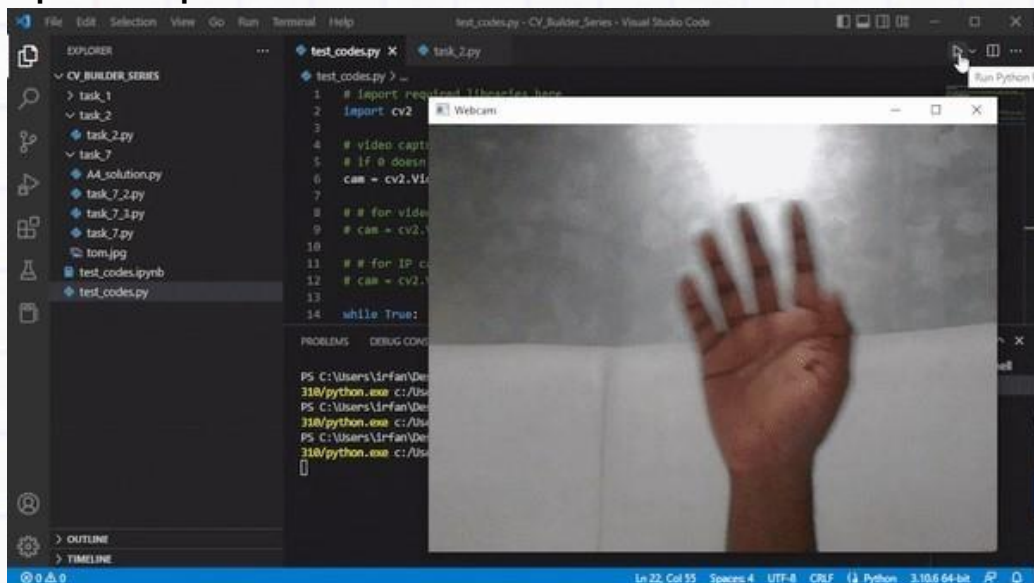
# # for video file, use:
# cam = cv2.VideoCapture('video_file_path')

# # for IP camera, use:
# cam = cv2.VideoCapture('IP_Address')

while True:
    _, frame = cam.read() # reading one frame from the camera object
    cv2.imshow('Webcam', frame) # display the current frame in a window
    named 'Webcam'

    # Waits for 1ms and check for the pressed key
    if cv2.waitKey(1) & 0xff == ord('q'): # press q to quit the camera (get
    out of loop)
        break
    cam.release() # close the camera
    cv2.destroyAllWindows() # Close all the active windows
```

### Expected Output:



## GETTING VIDEO PROPERTIES:

- There are a few useful functions in OpenCV to get information about the input video feed such as width, height, and fps.
- Try using the `cv2.CAP_PROP_FRAME_WIDTH`, `cv2.CAP_PROP_FRAME_HEIGHT`, and `cv2.CAP_PROP_FPS` to get the width, height, and fps respectively. Here is a list of other properties that you can access:

### VideoCaptureProperties.

#### Syntax:

```
cam.get(video_capture_property)
```

- `video_capture_property` > this can be any property that you want to access from the list of properties given here: [VideoCaptureProperties](#).

- Below is an example code that prints out the width, height, and frames per second of the camera feed.

#### Example Code:

```
# import required libraries here
import cv2

# video capture object where 0 is the camera number for a usb camera (or
# webcam)
# if 0 doesn't work, you might need to change the camera number to get the
# right camera you want to access
cam = cv2.VideoCapture(0)
# # for video file, use:
# Get the video file path
width = cam.get(cv2.CAP_PROP_FRAME_WIDTH)
height = cam.get(cv2.CAP_PROP_FRAME_HEIGHT)
# cam = cv2.VideoCapture('IP Address')
fps = cam.get(cv2.CAP_PROP_FPS)

while True:
    i, frame = cam.read() # reading one frame from the camera object
    cv2.imshow('Webcam', frame) # display the current frame in a window
    named 'Webcam'
    print('resolution:', width, 'x', height, '| frames per second:', fps)
    # Waits for 1ms and check for the pressed key
    if cv2.waitKey(1) & 0xff == ord('q'): # press q to quit the camera (get
    out of loop)
        break
    cam.release() # close the camera
    cv2.destroyAllWindows() # Close all the active windows
```

### Expected Output:

```
resolution: 640.0 x 480.0 | frames per second: 30.0  
resolution: 640.0 x 480.0 | frames per second: 30.0  
resolution: 640.0 x 480.0 | frames per second: 30.0  
resolution: 640.0 x 480.0 | frames per second: 30.0
```

## CHANGING VIDEO PROPERTIES:

- You can also set the video properties using `set` function which is similar to the `get` function in the above example.
- It takes in two arguments; the property and the value for it.

### Syntax:

```
cam.set(video_capture_property, value)
```

- `video_capture_property` > this can be any property that you want to set from the list of properties given here: [VideoCaptureProperties](#).
  - `value` > the value that you want that property to have
- Let's edit the previous example code in which, we set the width, height, and fps to a different value and the use `get` function, and print those values to check if it's been modified or not.

### Example Code:

```
# import required libraries here
import cv2

# video capture object where 0 is the camera number for a usb camera (or
webcam)
# if 0 doesn't work, you might need to change the camera number to get the
right camera you want to access
cam = cv2.VideoCapture(0)

# Changing video capture property
cam.set(cv2.CAP_PROP_FRAME_WIDTH,1280)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT,720)
cam.set(cv2.CAP_PROP_FPS,15)

# Getting camera feed width and height after modifying them
width = cam.get(cv2.CAP_PROP_FRAME_WIDTH)
height = cam.get(cv2.CAP_PROP_FRAME_HEIGHT)
fps = cam.get(cv2.CAP_PROP_FPS)

# # for video file, use:
# cam = cv2.VideoCapture('video_file_path')

# # for IP camera, use:
# cam = cv2.VideoCapture('IP_Address')

while True:
    i, frame = cam.read() # reading one frame from the camera object
    cv2.imshow('Webcam', frame) # display the current frame in a window
named 'Webcam'
    print('resolution:',width, 'x', height, '| frames per second:', fps)
    # Waits for 1ms and check for the pressed key
    if cv2.waitKey(1) & 0xff == ord('q'): # press q to quit the camera (get
out of the loop)
        break
    cam.release() # close the camera
    cv2.destroyAllWindows() # Close all the active windows
```

### Expected Output:

```
resolution: 1280.0 x 720.0 | frames per second: 30.0
```

- Notice how the resolution changed but the fps didn't. This is because it will only accept the properties supported by your camera.
- You can try other resolutions, if the camera supports the resolution, it will work, otherwise, it won't.

## ACCESSING AND MANIPULATING PIXELS:

- Here you will learn how to experiment with pixels by accessing and changing their values.
- Use the first example code from task\_1 to read an image. OpenCV reads images in numpy array format.
- In the code after reading the image print `image.shape` to check the **dimension** of the image. You could see something similar to the following:

```
print('image array dimension: ', image.shape)
```

**Output:** image array dimension: (451, 445, 3) > which means the image has 451 pixels widthwise, 445 pixels heightwise, and 3 channels (RGB).

- Now try accessing one of the pixels at the 5th row and 5th column position by `image[5,5]`.

```
print('a pixel: ', image[5,5])
```

**Output:** a pixel: [164 195 218] > here Blue, Green, and Red values are 164, 195, and 218 respectively for this "5th, 5th" pixel.

- Now let's take a small region from the image; `new_image = image[:10,:10]` (top left corner with 10x10 pixels) and try to split the channels with; `B = new_image[:, :, 0]`, `G = new_image[:, :, 1]` and `R = new_image[:, :, 2]`
- Add this code snippet to achieve the above task:

### Example Code Snippet:

```
new_image = image[:10,:10]
B = new_image[:, :, 0]
G = new_image[:, :, 1]
R = new_image[:, :, 2]

print('Blue Channel')
print(B)
print('Green Channel')
print(G)
print('Red Channel')
print(R)
```

## Expected Output:

```
Blue Channel
[[168 167 164 162 161 158 155 152 150 147]
 [167 167 164 163 162 159 156 155 152 149]
 [168 168 165 164 163 160 157 156 153 150]
  [170 169 168 165 164 162 159 156 153 152]
  [173 170 168 165 164 162 159 156 154 153]
  [172 171 168 167 164 164 162 160 158 155]
 [174 172 171 168 165 165 163 161 158 155]
 [175 174 172 171 168 165 164 162 159 156]
  [173 173 172 172 169 166 165 163 160 158]
  [173 173 172 172 169 168 165 164 161 159]]
Green Channel
[[198 197 195 194 191 191 189 188 185 185]
 [197 197 195 194 193 192 190 189 187 187]
 [198 198 196 195 194 193 191 190 189 188]
 [200 199 197 196 195 193 192 190 189 188]
 [201 200 198 196 195 193 192 191 191 189]
  [200 199 198 197 195 195 193 193 193 192]
 [203 201 199 198 196 196 195 195 194 192]
 [204 203 201 199 198 196 195 194 193 192]
  [202 202 201 200 199 198 196 196 194 194]
  [202 202 201 201 199 198 196 197 193 193]]
Red Channel
[[217 216 218 217 218 217 219 218 218 217]
 [216 216 216 217 218 218 219 219 220 219]
 [217 217 217 218 219 219 220 219 219 220]
  [219 218 218 217 218 218 218 219 219 218]
  [218 217 217 217 218 218 218 217 219 219]
 [217 216 215 216 216 218 218 219 219 220]
 [218 216 216 217 217 219 218 219 218 218]
  [218 217 216 216 217 217 218 217 217 216]
  [216 216 216 217 218 217 217 216 217 218]
  [216 216 216 216 218 217 217 217 216 216]]
```

- Same task can also be achieved by using the `split` function of OpenCV:

```
B,G,R = cv2.split(new_image)
```

- The `merge` function is used in a similar manner to merge individual channels.

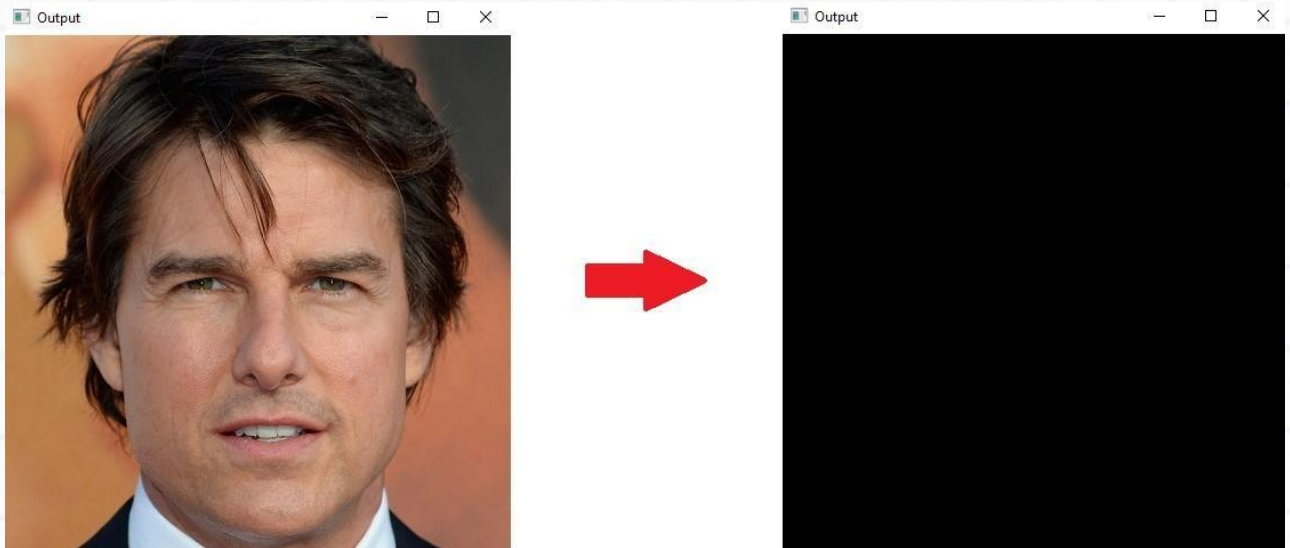
```
image_merged = cv.merge((b,g,r))
```

- We can also change the original value of pixels to something else. Try resetting all the pixel values of the input image to (0,0,0). By doing so, you can turn the entire image to black.

```
image[:, :] = (0,0,0)
```

- Here, : means all rows / all columns.

### Expected Output:



- You can experiment around with this by only changing the color of a small square at the corner or at the center of the frame. Play around and get comfortable with accessing and manipulating pixels.

## SAVING THE VIDEO:

- We can reuse the first "Example Code" and save the webcam feed.
- To save the video we can create an `output` object using the function `VideoWriter`

### Syntax:

```
output = cv2.VideoWriter(file_name, codec_code, fps, resolution)
```

- `file_name` > this can be just the name of the file or path+name.
- `codec_code` > 4 character code of codec used for compressing the frames. If given a -1 value. the program will print out a list of codec codes that can be used. `fps` > video frames per second.
- `resolution` > size of a frame in the video/ video resolution.

`output.write(frame)` > writes frame to the video file in every loop •

Try out the sample code below to save a video using OpenCV:

### Example Code:

```
# import required libraries here
import cv2

# video capture object where 0 is the camera number for a usb camera (or
webcam)
# if 0 doesn't work, you might need to change the camera number to get the
right camera you want to access
cam = cv2.VideoCapture(0)

# # for video file, use:
# cam = cv2.VideoCapture('video_file_path')

# # for IP camera, use:
# cam = cv2.VideoCapture('IP_Address')

width = int(cam.get(cv2.CAP_PROP_FRAME_WIDTH)) # convert to integer
height = int(cam.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cam.get(cv2.CAP_PROP_FPS)

output_file = './task_2/recording.MP4' # file location + name
output = cv2.VideoWriter(output_file, cv2.VideoWriter_fourcc(*'MJPG'), fps,
(width, height))

while True:
    _, frame = cam.read() # reading one frame from the camera object

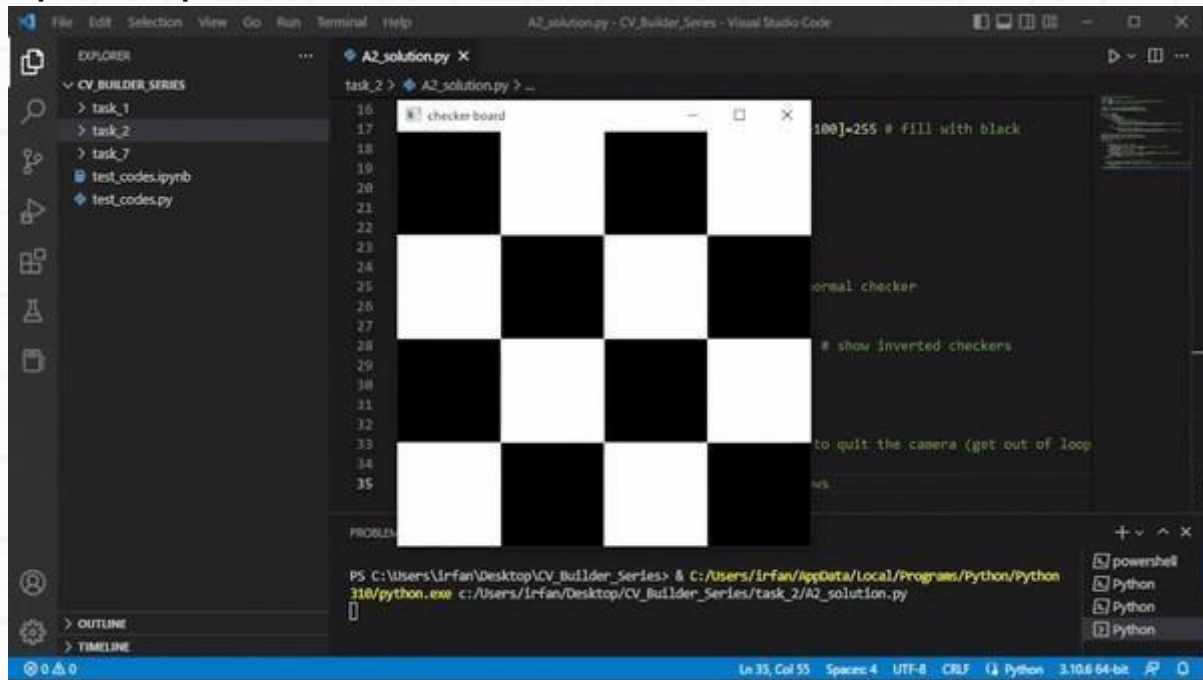
    cv2.imshow('Webcam', frame) # display the current frame in a window
named 'Webcam'
    output.write(frame)

    # Waits for 1ms and check for the pressed key
    if cv2.waitKey(1) & 0xff == ord('q'): # press q to quit the camera (get
out of the loop)
        break
cam.release() # close the camera
output.release() # close video writer
cv2.destroyAllWindows() # Close all the active windows
```

## WARMUP\_EXERCISE\_2:

- Create a 4x4 checkerboard with black and white colors, and then create a video where the checkerboard inverts color every second.
- Hint: `cv2.bitwise_not()` function might be useful to invert the checkerboard.

### Expected Output:



## 6. Drawing Functions:

**Objective:** In this section, you will learn how to draw objects like lines, rectangles, circles, text, etc. on an image.

- Create a `task_3.py` inside the "task\_3" folder and import `cv2` and `numpy`. Use the `line` function to create a line between two points.
- Let's start with a line by joining two points and then move on to other drawing functions. First, create a 300x300 pixel black background on which you will be drawing.

### Syntax:

```
cv2.line(image, start_point, end_point, color, thickness)
```

- `image` > input image or current frame.
- `color` > in (B, G, R) format.
- `thickness` > integer value, if -1 the color fills the object.

```
cv2.arrowedLine(image, start_point, end_point, color, thickness)
```

```
cv2.polylines(image, [points], isClosed, color, thickness)
```

- `[points]` > list of points to be joined.
- `isClosed` > if `True` forms a closed shape, `False` for an open shape.

```
cv2.rectangle(image, top_left, bottom_right, color, thickness)
```

- `top_left` > coordinates for the top left corner of the rectangle.
- `bottom_right` > coordinates for the bottom right corner of the rectangle.

```
cv2.circle(image, center, radius, color, thickness)
```

- `center` > coordinate for the center of the circle.
- `radius` > length of the radius (unit - pixels)

```
cv2.putText(image, text, bottom_left, font, thickness, color)
```

- `bottom_left` > bottom left corner point of the text.
- `font` > for list of fonts: **HersheyFonts**

- Let's take the following points to test these drawing functions:

```
p1 = [100,100]
p2 = [200,200]
p3 = [200,100]
p4 = [100,200]
```

- Try out the following drawing function examples and verify the outputs:

```
cv2.line(image,p1,p2,(0,255,0),2)
```

```
cv2.arrowedLine(image,p1,p2,(0,255,0),2)
```

```
cv2.polylines(image,[points],False,(0,255,0),2)
```

```
cv2.rectangle(image,p1,p2,(0,255,0),2) cv2.circle(circle,(150,150),50,(0,255,0),2)
```

```
cv2.putText(text,'sample_text', p4, cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0,255,0))
```

### Example Code:

```
# import required libraries
import cv2
import numpy as np

# Creating 6 different 300x300 pixel black canvas
line = np.zeros((300,300,3),dtype=np.uint8)
arrow = line.copy()
polyLine = line.copy()
rectangle = line.copy()
circle = line.copy()
text = line.copy()

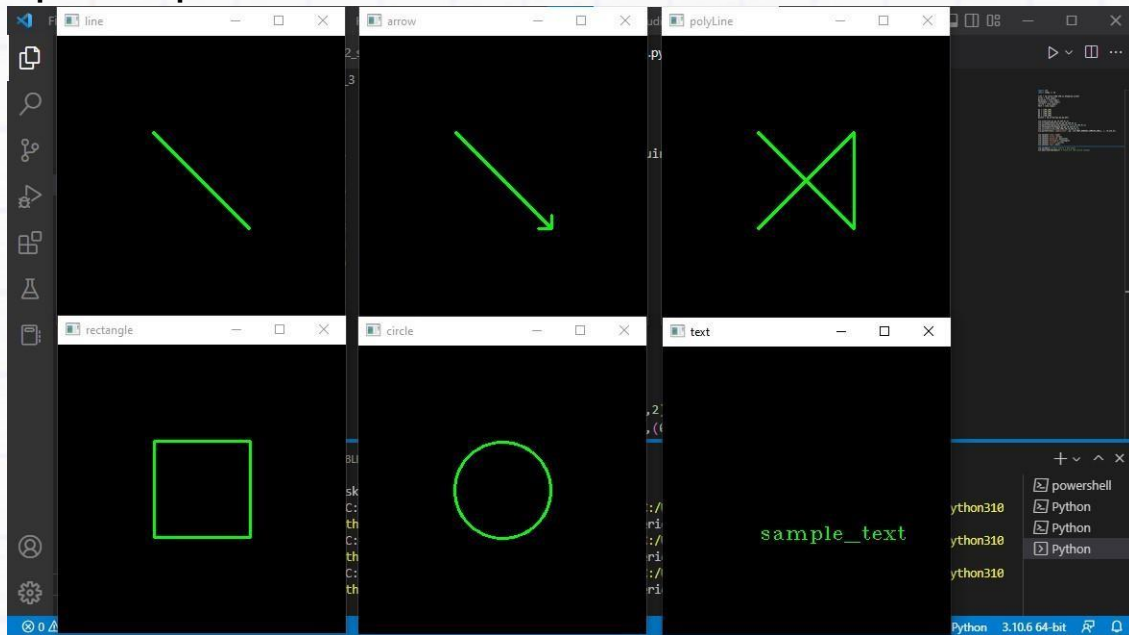
# Test points
p1 = [100,100]
p2 = [200,200]
p3 = [200,100]
p4 = [100,200]
points = np.array([p1,p2,p3,p4])

# Drawing functions
cv2.line(line,p1,p2,(0,255,0),2)
cv2.arrowedLine(arrow,p1,p2,(0,255,0),2)
cv2.polylines(polyLine,[points],False,(0,255,0),2)
cv2.rectangle(rectangle,p1,p2,(0,255,0),2)
cv2.circle(circle,(150,150),50,(0,255,0),2)
cv2.putText(text,'sample_text', p4, cv2.FONT_HERSHEY_COMPLEX_SMALL, 1,
(0,255,0))

# Display the drawn objects
cv2.imshow('line',line)
cv2.imshow('arrow',arrow)
cv2.imshow('polyLine',polyLine)
cv2.imshow('rectangle',rectangle)
cv2.imshow('circle',circle)
cv2.imshow('text',text)

cv2.waitKey() # Wait untill a key press
cv2.destroyAllWindows() # Close all the active windows
```

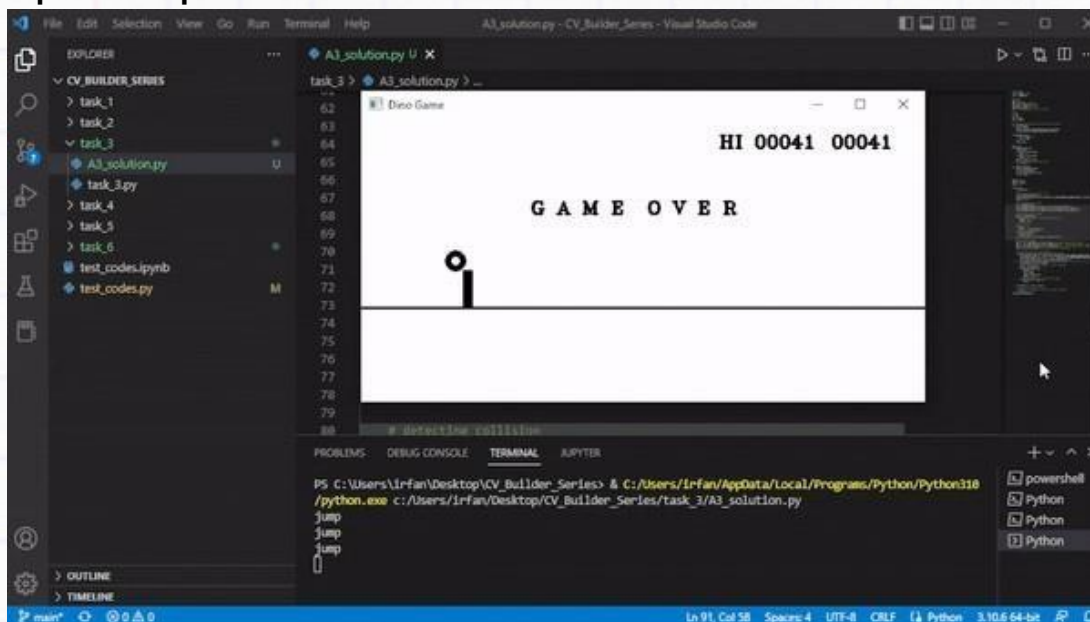
### Expected output



### WARMUP\_EXERCISE\_3:

- Create a basic version of chrome’s dino game. You don’t have to build an exact copy of it.
- A simple ball for dino, rectangular bars for the obstacles that move towards the ball, and then the ball jumps when you press space. (use `waitKey` along with space bar detection, just like we do for quitting the program when you press ‘q’)
- Score on the top right corner and the game should stop and display ‘Game Over’ when you hit an obstacle.

### Expected Output:



## 7. Interacting with the video:

**Objective:** In this section, you will learn how to interact with the frame/window with mouse events and trackbars.

### MOUSE EVENTS:

- Create a folder named "task\_4" inside "CV\_Builder\_Series".
- Create a new file named `task_4.py` inside the folder "task\_4". And import `cv2` and `numpy`.
- Create a mouse callback function that is executed when a mouse event takes place. It gives information like the position of the mouse, the type of event, and useful flags. Now using `setMouseCallback` function that takes in the window name and a callback function as parameters we can have access to that information.

#### Syntax

```
function_name(event,x_position,y_position,flags,parameters)
cv2.setMouseCallback('window_name',function_name)
```

#### Event Flags

```
EVENT_FLAG_LBUTTONDOWN = 1
EVENT_FLAG_RBUTTONDOWN = 2
EVENT_FLAG_MBUTTONDOWN = 4
EVENT_FLAG_CTRLKEY = 8
EVENT_FLAG_SHIFTKEY = 16
EVENT_FLAG_ALTKEY = 32
```

#### Event Types

```
EVENT_MOUSEMOVE = 0
EVENT_LBUTTONDOWN = 1
EVENT_RBUTTONDOWN = 2
EVENT_MBUTTONDOWN = 3
EVENT_LBUTTONUP = 4
EVENT_RBUTTONUP = 5
EVENT_MBUTTONUP = 6
EVENT_LBUTTONDBLCLK = 7
EVENT_RBUTTONDBLCLK = 8
EVENT_MBUTTONDBLCLK = 9
EVENT_MOUSEWHEEL = 10
EVENT_MOUSEHWHEEL = 11
```

## Example

```
# importing required libraries
import cv2
import numpy as np

# Mouse callback function
def mouseClicked(event, xPos, yPos, flags, param):
    print(event, xPos, yPos, flags, param)

# Creating a black image/frame (0 pixel value) of 500x500 size
frame = np.zeros((500,500), np.uint8)

# Creating an window to display image/frame
cv2.namedWindow('FRAME')

# This function detects every new events and triggers the "mouseClick"
function
cv2.setMouseCallback('FRAME',mouseClick)

while True:
    cv2.imshow('FRAME',frame)
    if cv2.waitKey(1) & 0xff == ord('q'): # to quit press 'q'
        break
cv2.destroyAllWindows()
```

**The above code prints out the following event information:**

```
0 118 214 0 None
0 123 210 0 None
0 131 205 0 None
0 137 200 0 None
0 143 196 0 None
...
0 196 212 0 None
0 198 210 0 None
0 198 209 0 None
1 198 209 1 None
4 198 209 0 None
```

## DRAW A RECTANGLE WITH MOUSE

- Let's draw a rectangle by dragging the mouse, on left click press, and stop on release. For this we need a few global variables initialized as follows:

```
# Global variables shared between the mouseClicked function and rest of the
code
draw = False
p1 = (0,0) # top left corner point
p2 = p1 # bottom right corner point
```

- Mouse click can be detected by the event `cv2.EVENT_LBUTTONDOWN`, movement by `cv2.EVENT_MOUSEMOVE`, and release by `cv2.EVENT_LBUTTONUP` and get the pointer position.
- An example code with a detailed explanation is given below:

**Example Code:** [task\\_4.py](#) (NEXT PAGE)

```

# importing required libraries
import cv2
import numpy as np

# Global variables shared between the mouseClicked function and rest of the
code
draw = False
p1 = (0,0) # top left corner point
p2 = p1 # bottom right corner point

# Mouse callback function
def mouseClicked(event,xPos,yPos,flags,param):
    # print(event,xPos,yPos,flags,param)

    # Global variables shared between the mouseClicked function and rest of
the code
    global draw,p1,p2

    # if left click press event, start drawing with p1 as top left corner
point coordinates
    if event==cv2.EVENT_LBUTTONDOWN:
        draw = True
        p1 = (xPos,yPos)
        p2 = p1
        # Continuously update bottom right corner point (p2) of rectangle on
mouse move event
        if event==cv2.EVENT_MOUSEMOVE and draw:
            p2 = (xPos,yPos)
        # if left click release, stop drawing
        if event==cv2.EVENT_LBUTTONUP:
            draw = False

# Creating a black image/frame (0 pixel value) of 500x500 size
frame = np.zeros((500,500,3), np.uint8)

# Creating an window to display image/frame
cv2.namedWindow('FRAME')

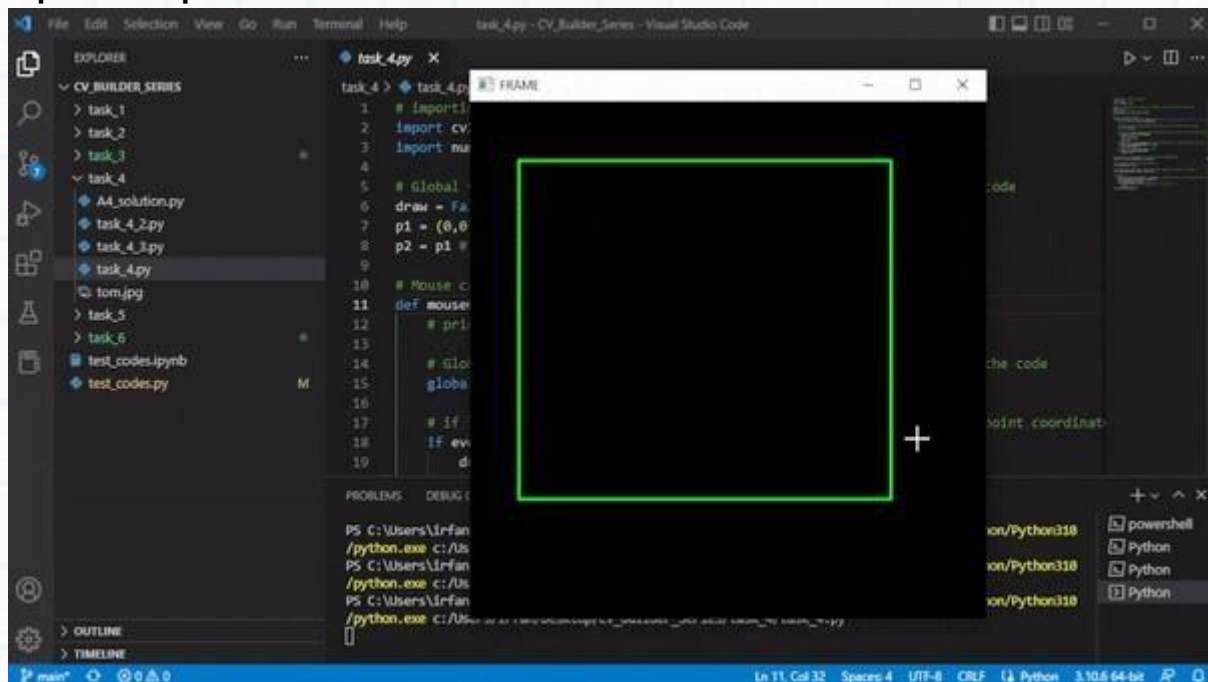
# This function detects every new events and triggers the "mouseClick"
function
cv2.setMouseCallback('FRAME',mouseClick)

while True:
    frame = np.zeros((500,500,3), np.uint8) # renew black frame in every
loop (this simulates a video)
    cv2.rectangle(frame,p1,p2,(0,255,0),2)
    cv2.imshow('FRAME',frame)
    if cv2.waitKey(1) & 0xff == ord('q'): # to quit press 'q'

```

```
break
cv2.destroyAllWindows()
```

### Expected Output:



## DRAW A CURVE USING A MOUSE

- ◆ Let's now draw a curve with mouse click drag and stop drawing on left click release. Also, the frame resets back to black on the right click. We will be using the `cv2.line` function for creating the curve. As the points will be very close to each other, the chain of line segments will look like a curve.
- ◆ For this task, we can reuse `task_4.py` and make some changes to it. Create a python file named `task_4_2.py` and copy `task_4.py` code in it.
- ◆ We need a new global variable `reset` that erases the frame/canvas on the right click. And during the event of left click and drag, we collect the points in `p1`, `p2` accordingly and keep drawing lines between them.
- ◆ An example code with a detailed explanation is given below:

### Example Code:

```
# importing required libraries
import cv2
import numpy as np

# Global variables shared between the mouseClicked function and rest of the
code
draw = False
reset = False
# initially p1 and p2 = 0
p1 = (0,0) # First point of line segment
```

(NEXT PAGE)

```

p2 = p1 # Second point of line segment

# Mouse callback function
def mouseClicked(event, xPos, yPos, flags, param):
    # print(event, xPos, yPos, flags, param)

    # Global variables shared between the mouseClicked function and rest of
the code
    global draw, reset, p1, p2

    # if left click press event, enable drawing and p1 and p2 as current
position
    if event==cv2.EVENT_LBUTTONDOWN:
        draw = True
        reset = False
        p1 = (xPos, yPos)
        p2 = p1
        # Continuously update p2 on mouse movement and left mouse press
        if event==cv2.EVENT_MOUSEMOVE and draw:
            p2 = (xPos, yPos)
        # if left click release, stop drawing
        if event==cv2.EVENT_LBUTTONUP:
            draw = False
        # if right click, reset the frame/canvas, and p1 and p2 to 0
        if event==cv2.EVENT_RBUTTONDOWN:
            reset = True
            p1 = (0, 0)
            p2 = p1

# Creating a black image/frame (0 pixel value) of 500x500 size
frame = np.zeros((500, 500, 3), np.uint8)

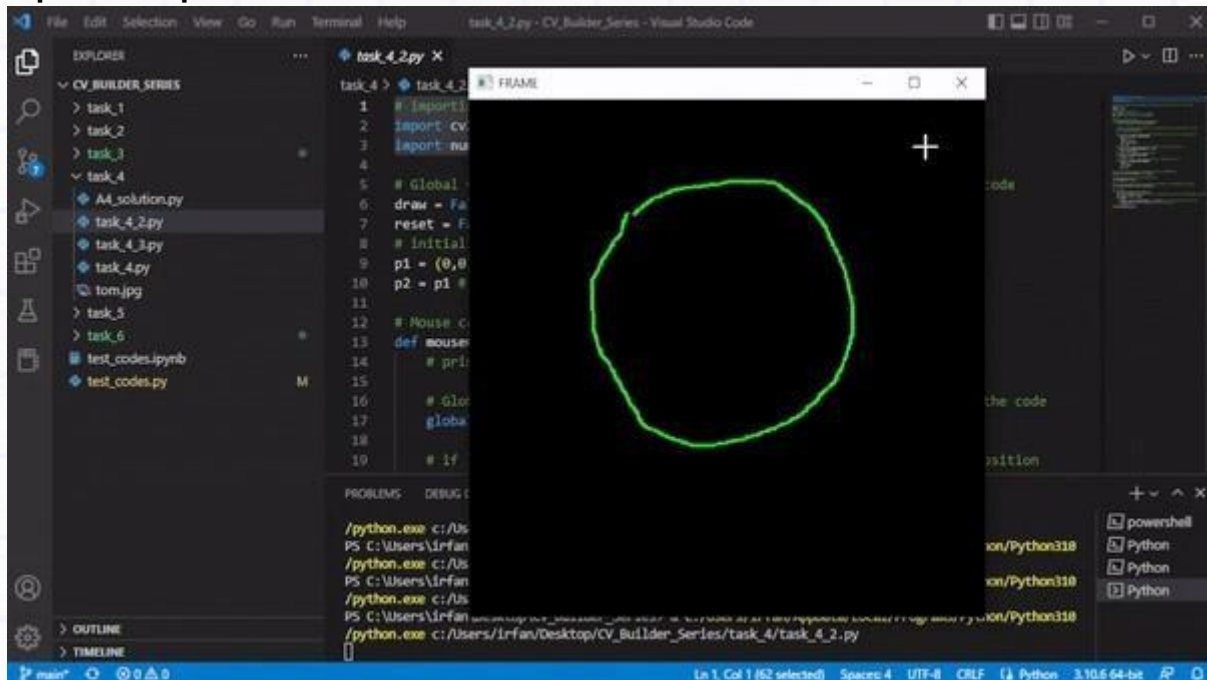
# Creating an window to display image/frame
cv2.namedWindow('FRAME')

# This function detects every new events and triggers the "mouseClick"
function
cv2.setMouseCallback('FRAME', mouseClicked)

while True:
    cv2.line(frame, p1, p2, (0, 255, 0), 2)
    p1 = p2 # swapping points for next line segment (p2 copies to p1 and p2
updates to latest position)
    cv2.imshow('FRAME', frame)
    if reset:
        frame = np.zeros((500, 500, 3), np.uint8) # renew black frame on right
click
    if cv2.waitKey(1) & 0xff == ord('q'): # to quit press 'q'
        break
cv2.destroyAllWindows()

```

## Expected Output:



## INTERACTING WITH TRACKBARS IN OPENCV

- Trackbars in OpenCV allow us to change a variable value while the program is still running. This will be helpful when we need to change some variables and see their effects on the output in real time without closing and relaunching the program.
- This process is similar to mouse events. To create a trackbar we will be using the `cv2.createTrackbar()` function which takes in a few arguments along with a `callback_function`

### Syntax:

`callback_function(value)`

- `value` updates when the slider moves.

`cv.createTrackbar(trackbar_name, window_name, slider_initial_position, slider_max_length, callback_function)`

- Create a python file named `task_4_3.py` and try out the following example code that prints out the trackbar position values.

### Example:

```
# importing required libraries
import cv2
import numpy as np

# Variables for trackbar value
val = 0

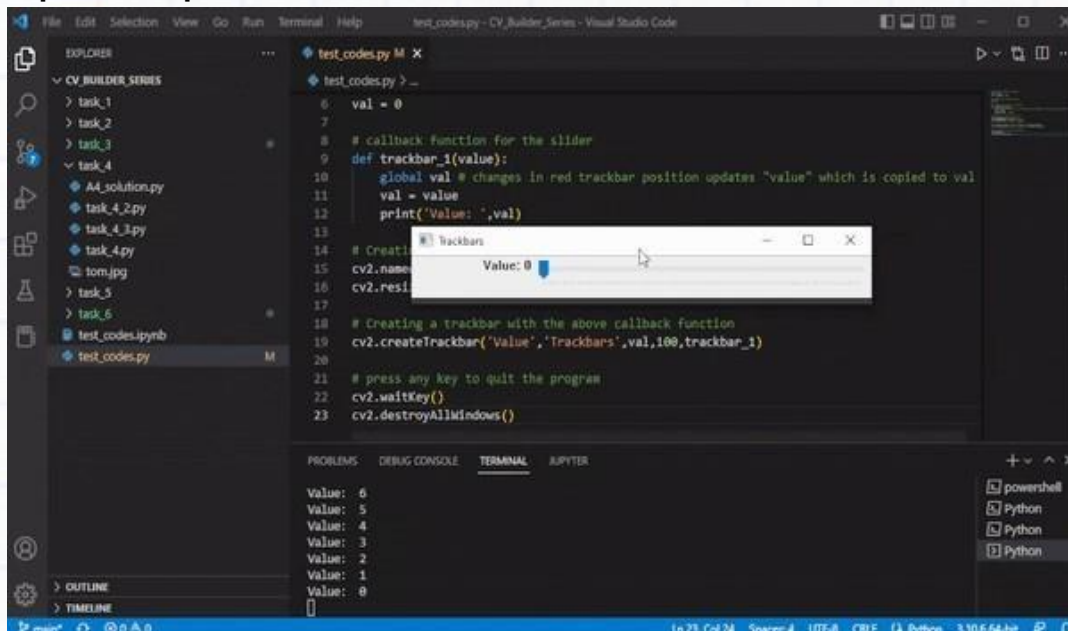
# callback function for the slider
def trackbar_1(value):
    global val # changes in red trackbar position updates "value" which is
    copied to val
    val = value
    print('Value: ',val)

# Creating an window for trackbar
cv2.namedWindow('Trackbars')
cv2.resizeWindow('Trackbars',500,50)

# Creating a trackbar with the above callback function
cv2.createTrackbar('Value', 'Trackbars',val,100,trackbar_1)

# press any key to quit the program
cv2.waitKey()
cv2.destroyAllWindows()
```

### Expected Output:



- Let's now write a program that individually tweaks the Red, Green, and Blue channels of an image using a slider, and create a color mixer.

- Similar to mouse events, we can use a global variable to access the trackbar value and modify a blank frame's RGB channel according to the trackbar values to create a color mixer.
- Tryout the following code with detailed comments to create a color mixer:

**Example:**

```

# importing required libraries
import cv2
import numpy as np

# Variables for RGB channel values (initially 0)
R = 0
G = 0
B = 0

# callback function for RED slider
def redValue(value):
    global R # changes in red trackbar position updates "value" which is
    copied to R
    R = value
    # print('Red Value: ',R)

# callback function for GREEN slider
def greenValue(value):
    global G # changes in green trackbar position updates "value" which is
    copied to G
    G = value
    # print('Green Value: ',G)

# callback function for BLUE slider
def blueValue(value):
    global B # changes in blue trackbar position updates "value" which is
    copied to B
    B = value
    # print('Blue Value: ',B)

# Creating a black image/frame (0 pixel value) of 500x500 size
frame = np.zeros((500,500,3), np.uint8)

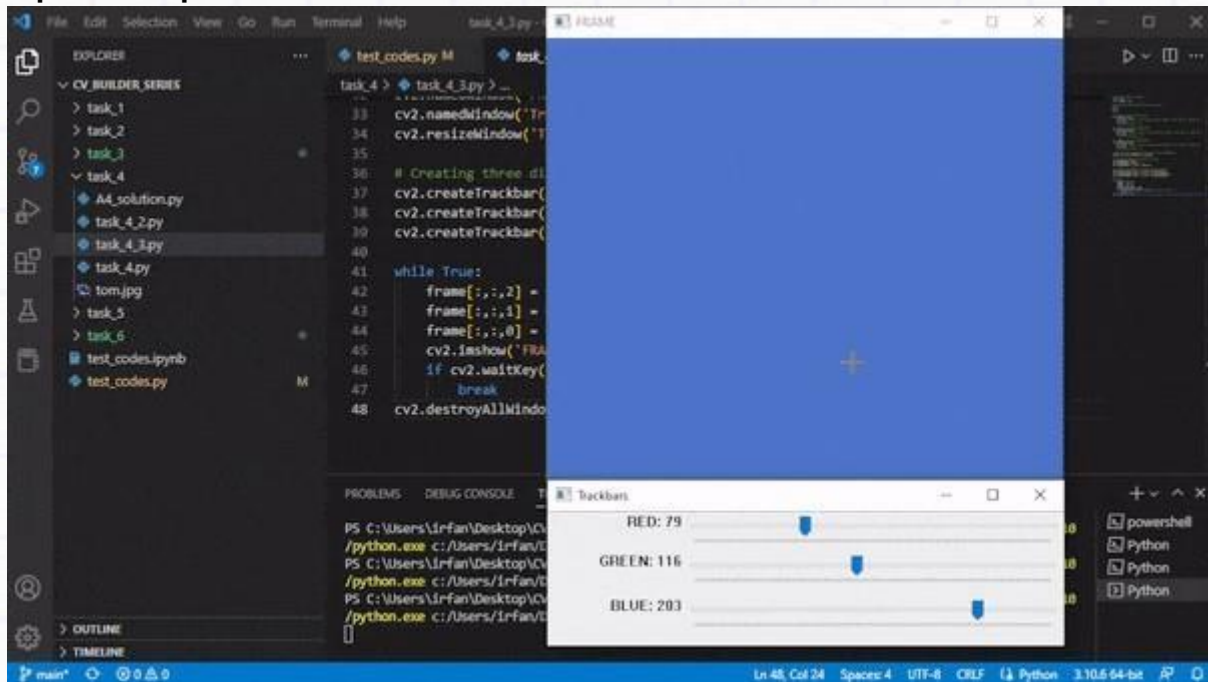
# Creating an window for trackbars and frame
cv2.namedWindow('FRAME')
cv2.namedWindow('Trackbars')
cv2.resizeWindow('Trackbars',500,130)

# Creating three different trackbars for RGB channels
cv2.createTrackbar('RED','Trackbars',R,255,redValue)
cv2.createTrackbar('GREEN','Trackbars',G,255,greenValue)
cv2.createTrackbar('BLUE','Trackbars',B,255,blueValue)

while True:
    frame[:, :, 2] = R
    frame[:, :, 1] = G
    frame[:, :, 0] = B
    cv2.imshow('FRAME',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'): # to quit press 'q'
        break
cv2.destroyAllWindows()

```

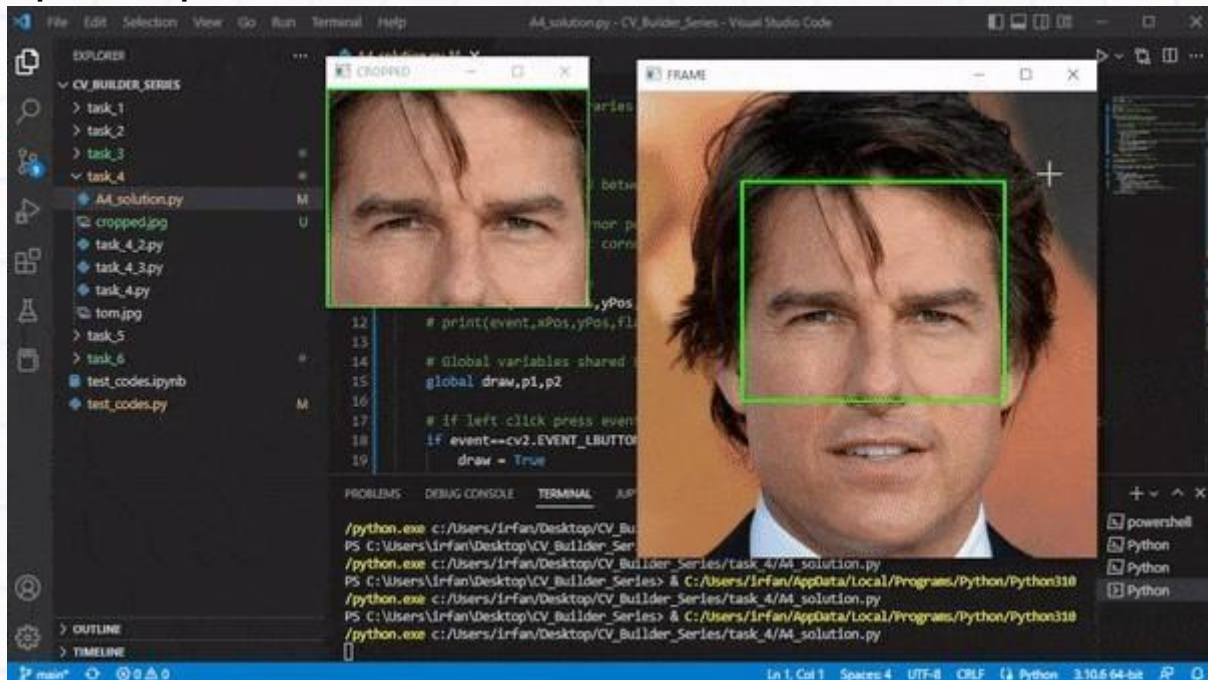
### Expected Output:



### WARMUP\_EXERCISE\_4:

- Create a simple program that can read an image, and crop a part of it by mouse click and drag (just like we did for drawing the rectangle), and on mouse release, it should save the cropped image in the current working directory.

### Expected Output:

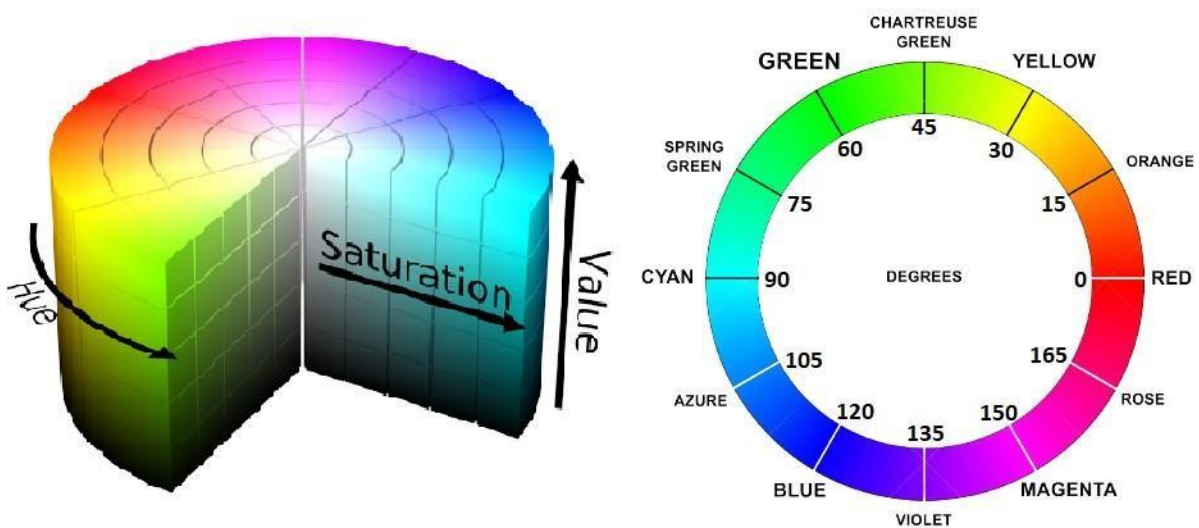


## 8. Object detection and tracking with a color mask.

**Objective:** In this section, you will learn how to select and mask a color, find contours around the masked color, and then create a bounding box around it for tracking. You will also learn how to detect and track anything using a template image of an interesting object.

### UNDERSTANDING HSV COLORSPACE

- Let's start by understanding the HSV color space. HSV stands for Hue, Saturation, and Value.
- HSV color space is different from RGB, where all the variables are responsible for a particular color combination, its darkness, etc. In HSV, color is represented by Hue value (0-180), saturation (0-255) varies from being white to full color, and value (0-255) varies from being dark to full color.



- Hence isolating a color becomes easy in HSV color space than the RGB.

### MASKING COLOR

- Create a python file `task_5.py` inside a folder named "task\_5".
- Create trackbars for; `hue_low`, `hue_high`, `sat_low`, `sat_high`, `val_low`, and `val_high`. These values will be useful for creating a `lower_bound` and `upper_bound` to isolate a color range. this isolation is achieved by `inRange` function that takes in HSV image, `lower_bound`, and `upper_bound` and returns a mask.
- We can use this mask to isolate the object of interest by bitwise ANDing the mask with the original image or to get contours around it and even bounding boxes for them.
- To extract the masked region from the original video, you might need to use `bitwise_and` function. For more information, refer to this documentation: [Arithmetic Operations on Images](#)

**Syntax:**

```
lowerBound = np.array([hueLow,satLow,valLow])  
upperBound = np.array([hueHigh,satHigh,valHigh])  
mask = cv2.inRange(frameHSV, lowerBound, upperBound)  
cv2.bitwise_and(image,image,mask=mask)
```

- Try the following program that masks out the blue color from an image and prints out the lower and upper bound range for the blue ball with the help of trackbars. These color-bound values will be useful later to track an object (in this case ball) using its color.

**Example Code: (NEXT PAGE)**

```

# import required libraries
import cv2
import numpy as np

#setting up callback functions for trackbars
def onTrack1(val):
    global hueLow
    hueLow = val

def onTrack2(val):
    global hueHigh
    hueHigh = val

def onTrack3(val):
    global satLow
    satLow = val

def onTrack4(val):
    global satHigh
    satHigh = val

def onTrack5(val):
    global valLow
    valLow = val

def onTrack6(val):
    global valHigh
    valHigh = val

# wTrackbar window
cv2.namedWindow('Trackbars')
cv2.resizeWindow('Trackbars',400,300)

#initial values of trackbar slider
hueLow = 0
hueHigh = 0
satLow = 0
satHigh = 0
valLow = 0
valHigh = 0

# reating trackbars
cv2.createTrackbar('Hue low','Trackbars',110,179,onTrack1)
cv2.createTrackbar('Hue High','Trackbars',150,179,onTrack2)
cv2.createTrackbar('Sat low','Trackbars',80,255,onTrack3)
cv2.createTrackbar('Sat High','Trackbars',255,255,onTrack4)
cv2.createTrackbar('Val low','Trackbars',134,255,onTrack5)
cv2.createTrackbar('Val High','Trackbars',255,255,onTrack6)

# Input Image
image = cv2.imread('./task_5/ball.png')

```

```

while True:

    frameHSV = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
    lowerBound = np.array([hueLow,satLow,valLow])#lower and upper boundary
for color range in HSV
    upperBound = np.array([hueHigh,satHigh,valHigh])
    mask = cv2.inRange(frameHSV, lowerBound, upperBound)#Creating Mask using
the color range
    masked = cv2.bitwise_and(image,image,mask=mask)

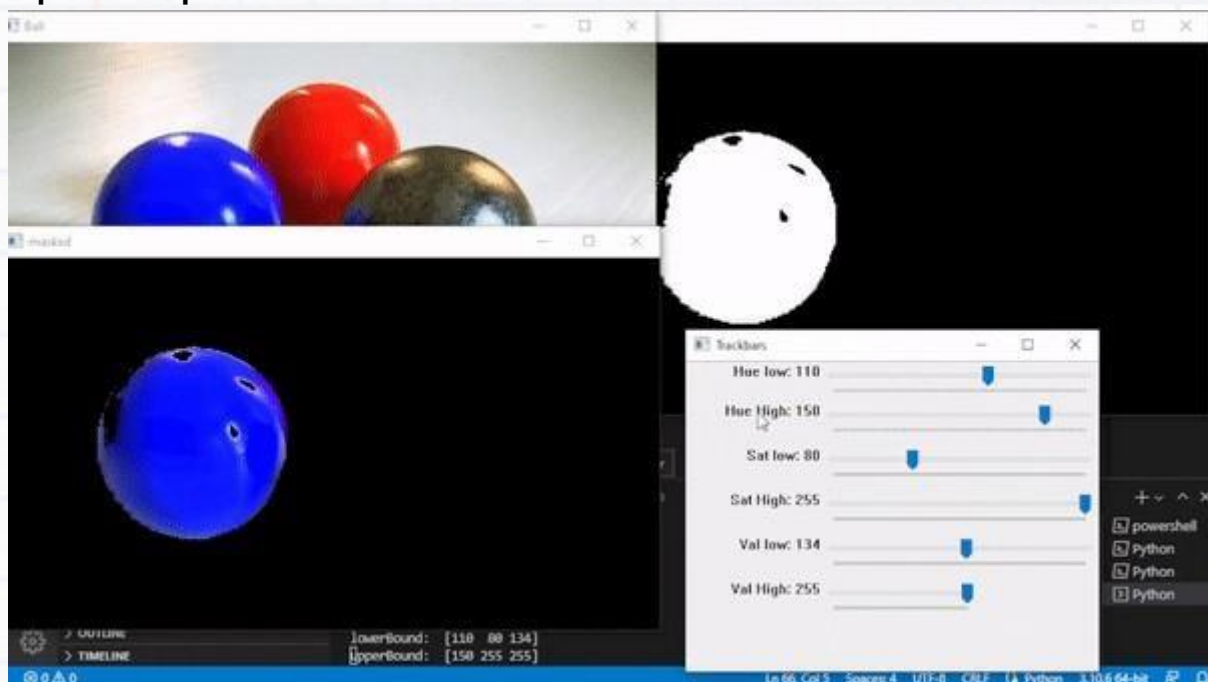
    cv2.imshow('mask', mask)
    cv2.imshow('Ball', image)
    cv2.imshow('masked',masked)

    print("lowerBound: ", lowerBound)
    print("upperBound: ", upperBound)
    if cv2.waitKey(1) & 0xff == ord('q'): # to quit the camera press 'q'
        break

cv2.destroyAllWindows()

```

### Expected Output:



## FINDING CONTOURS AND BOUNDING BOX:

- ◆ Since we now have the color bound for masking the blue color, we can now apply it on a video to mask the blue ball, find the contours around it, and then the bounding box to track the blue ball.
- ◆ `findContours` function is used to get contours from a binary image. A binary image only contains two values 0 for black and 1 for white.

### Syntax:

```
contours,hierarchy = cv2.findContours(image,retrieval_mode,approximation_method)
```

- `image` > image on which contours have to be found.
  - `retrieval_mode` > `cv.RETR_LIST`, `cv.RETR_TREE`, and `cv2.RETR_EXTERNAL` can be used based on how you want the contours to be presented. For more details refer: [RetrievalModes](#)
  - `approximation_method` > contour approximation method. Details on different approximation methods can be found here: [ContourApproximationModes](#)
  - `cv2.boundingRect(contour)` > this function is used to get a bounding box around the masked object. This is very useful in object tracking.
- ◆ Below is a sample code that tracks a blue ball in a video.

### Example Code: (NEXT PAGE)

```

# import required libraries
import cv2
import numpy as np

# Input video file
cam = cv2.VideoCapture('./task_5/ball.wmv')

while True:
    _, frame = cam.read()
    # converting to HSV for masking
    frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # lower and upper bound for color from last program
    lowerBound = np.array([110, 80, 134]) # lower and upper boundary for color
    range in HSV
    upperBound = np.array([150, 255, 255])
    mask = cv2.inRange(frameHSV, lowerBound, upperBound) # Creating Mask using
    the color range

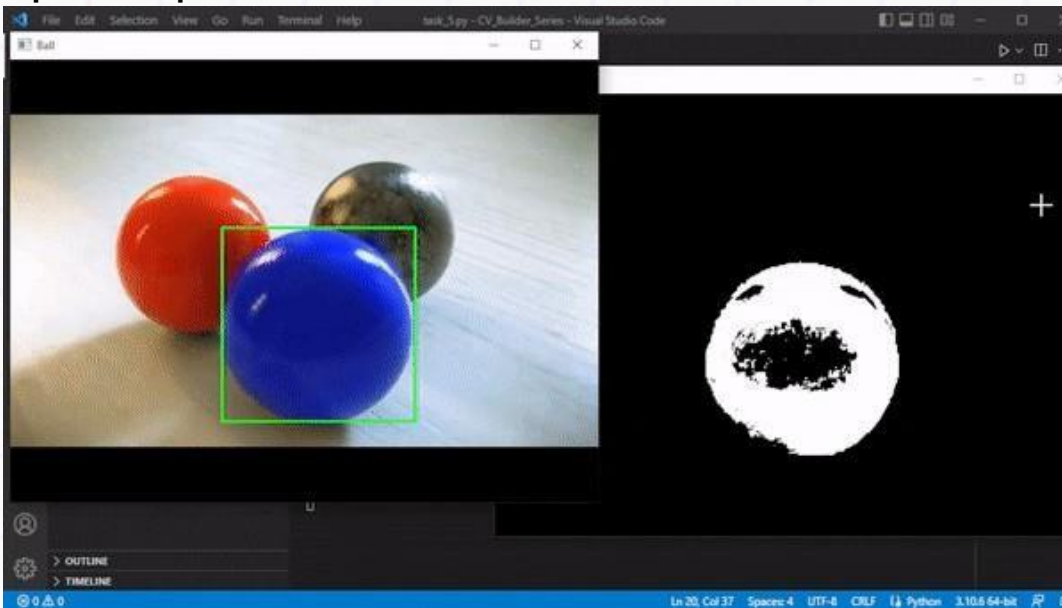
    # Finding contours on masked frame
    ballContours, hierarchy =
cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) # contours
around mask
    # If multiple set of contours available, iterate through each
    for ballContour in ballContours:
        area = cv2.contourArea(ballContour)
        if area > 500: # to filter out noise. This avoids very small contours
that could be noise
            x, y, w, h = cv2.boundingRect(ballContour) # this function returns
position and size of bounding box for tracking
            # print(x, y, w, h)
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2) # draw rectangle
around blue box

    cv2.imshow('mask', mask)
    cv2.imshow('Ball', frame)

    if cv2.waitKey(1) & 0xff == ord('q'): # to quit the camera press 'q'
        break
cam.release()
cv2.destroyAllWindows()

```

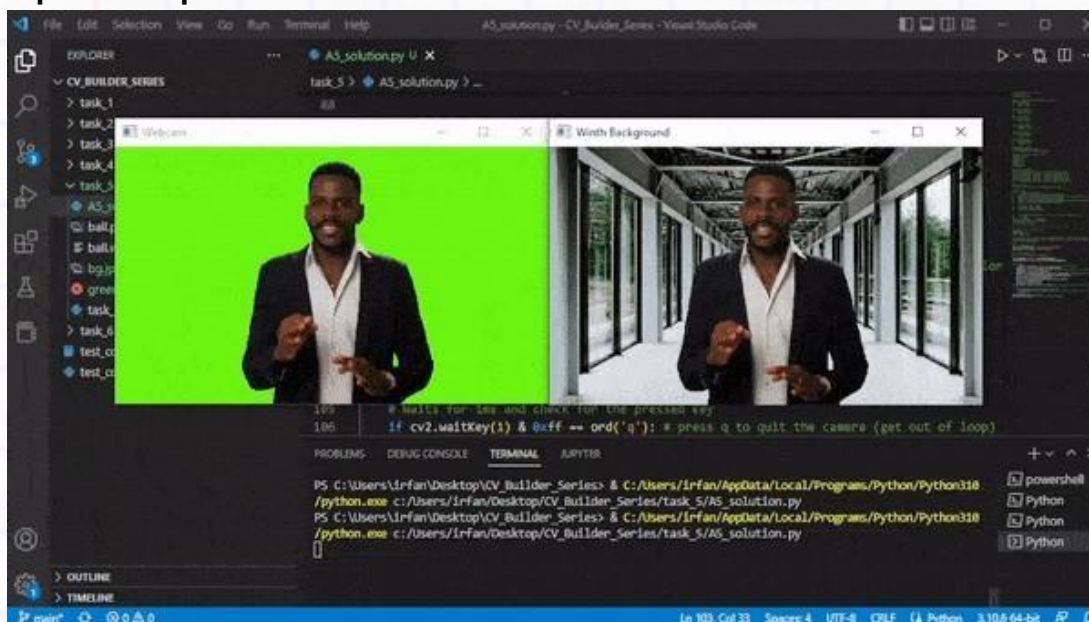
### Expected Output:



### WARMUP\_EXERCISE\_5:

- Write a program that removes the background (Green Screen) from the webcam feed and replace the background with other images, just like we have on zoom calls.
- Try putting a solid colored canvas behind you so that the color detection and masking become easy. (OR try with the given sample video)
- Refer to this documentation for arithmetic operations: [Arithmetic Operations on Images](#)
- NOTE: The resolution of the webcam feed/video and background should be the same.

### Expected Output:



## 9. Frame manipulation and transformation:

**Objective:** In this section, you will learn to transform and manipulate the frames and hence the video. You will be performing, resizing, rotating, finding edges, finding and drawing contours, smoothening and perspective warping.

### RESIZE:

- Create a python file `task_6.py` inside a folder `task_6`.
- Reuse the “**READING IMAGE FILE**” code from previous section to read an image and display it.
- Use `resize` function to reduce the dimension of the input image by half.

#### Syntax:

```
cv2.resize(image,(height,width))
```

- `(height,width)` > height and width of the resized image.

#### Example Code:

```
# Importing OpenCV Library
import cv2

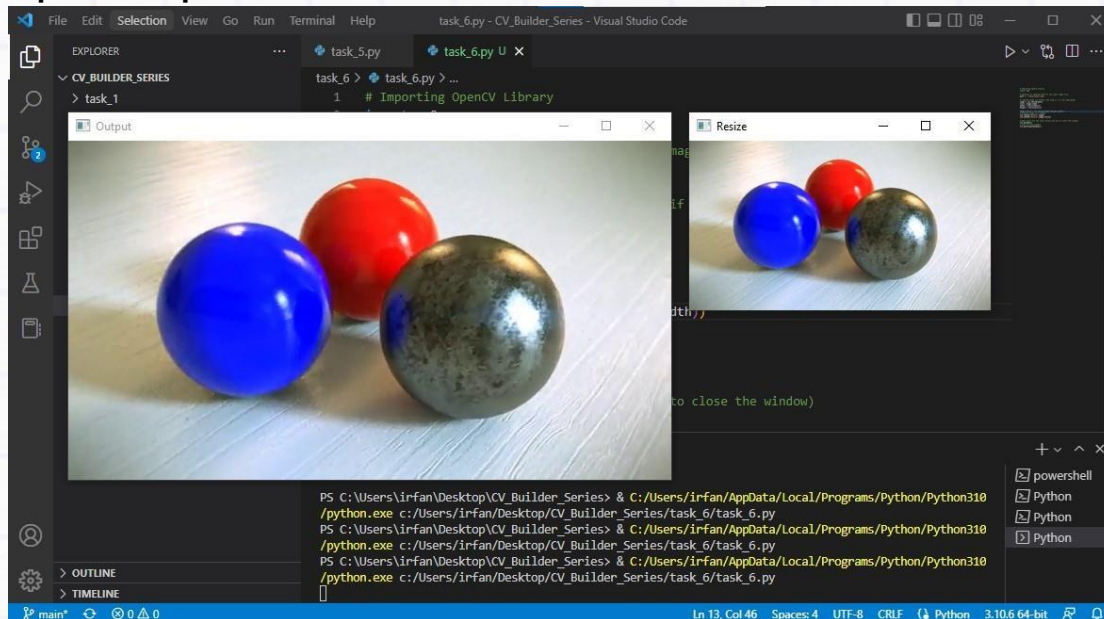
# Relative or absolute path of the input image file
path = "./task_6/ball.png"

# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)
size = image.shape
width = int(size[0]/2)
height = int(size[1]/2)

image_resize = cv2.resize(image,(height,width))
# Display image in a window
cv2.imshow("Output",image)
cv2.imshow("Resize",image_resize)

# Wait until any key press (press any key to close the window)
cv2.waitKey()
# kill all the windows
cv2.destroyAllWindows()
```

## Expected Output:



## ROTATE:

- To rotate an image use `rotate` function:

### Syntax:

`cv2.rotate(image, rotate_code)`; possible options for `rotate_code` are; `ROTATE_90_CLOCKWISE`, `ROTATE_180` and `ROTATE_90_COUNTERCLOCKWISE`

### Example Code:

```
# Importing OpenCV Library
import cv2

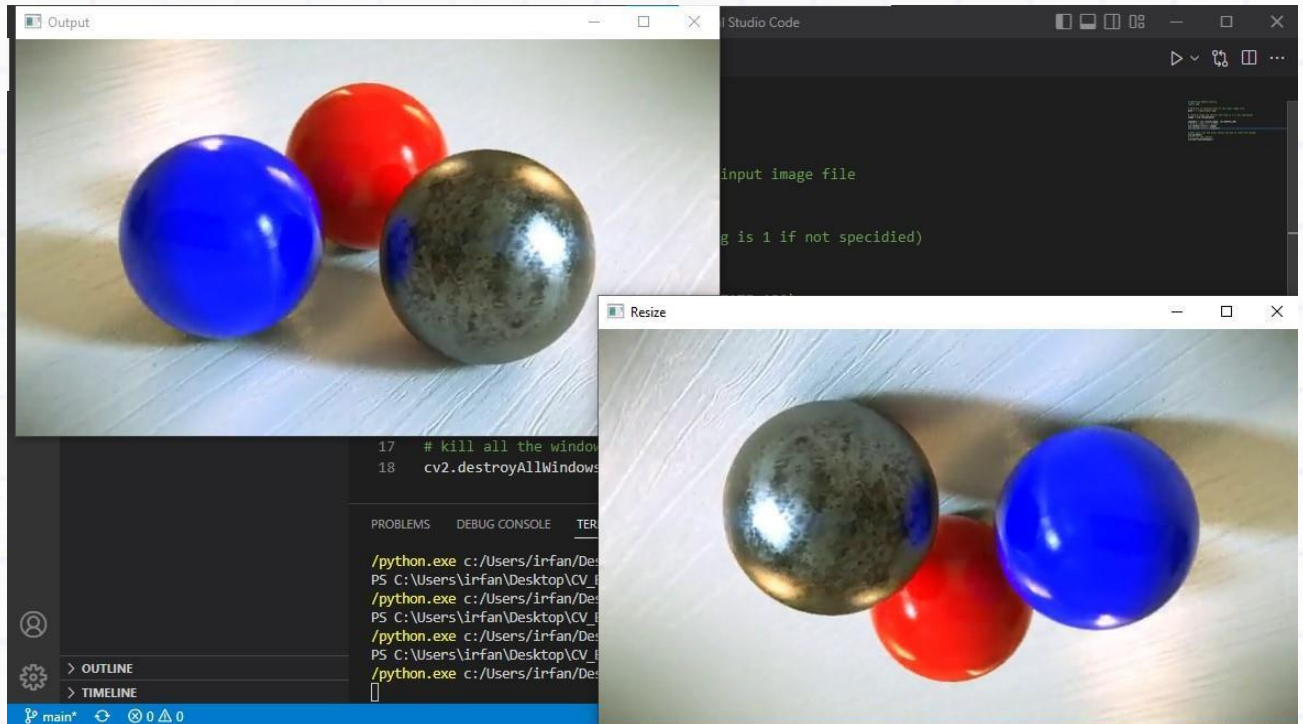
# Relative or absolute path of the input image file
path = "./task_6/ball.png"

# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)

imageRot = cv2.rotate(image, cv2.ROTATE_180)
# Display image in a window
cv2.imshow("Output", image)
cv2.imshow("Resize", imageRot)

# Wait until any key press (press any key to close the window)
cv2.waitKey()
# kill all the windows
cv2.destroyAllWindows()
```

## Expected Output:



- To rotate the image in a certain angle, `warpAffine` function is used with a rotation matrix created by `getRotationMatrix2D` function.

### Syntax:

```
M = cv2.getRotationMatrix2D(center_of_rotation, angle, scale)
```

- o `scale` > zooming scale `rotated_image= cv2.warpAffine(image, M, (height, width))`

### Example Code: (NEXT PAGE)

```

# Importing OpenCV Library
import cv2

# Relative or absolute path of the input image file
path = "./task_6/ball.png"

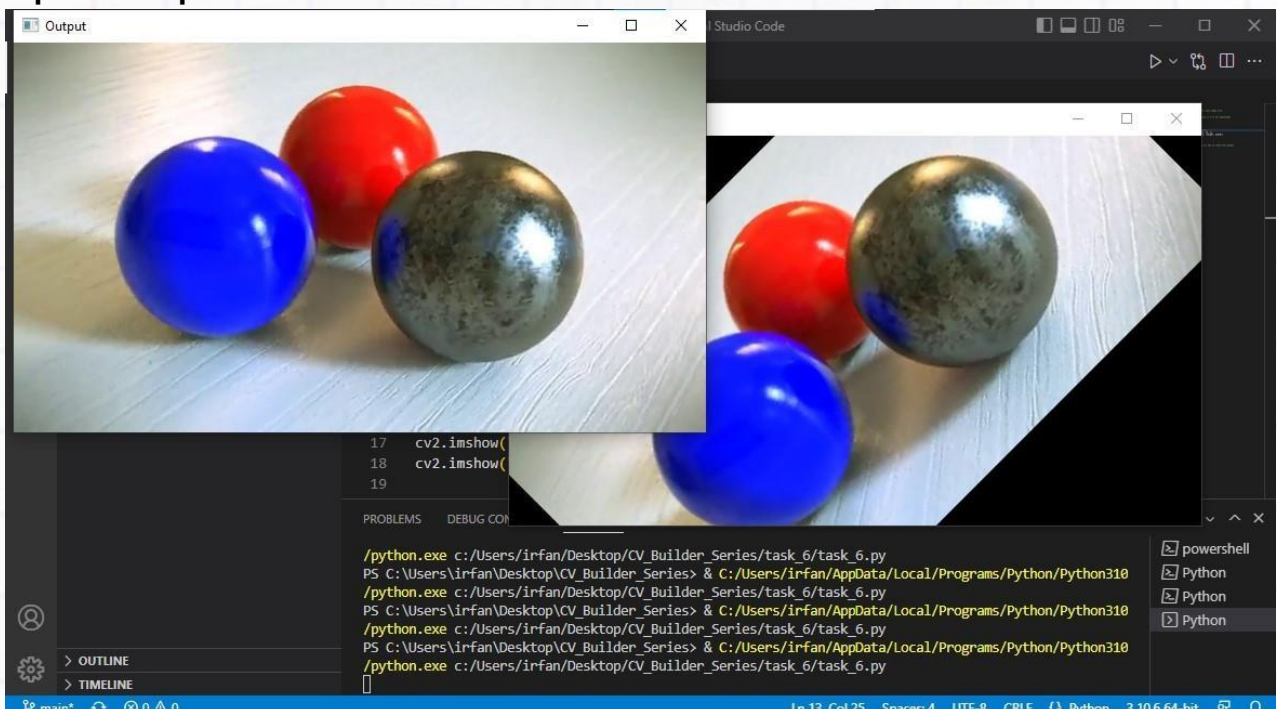
# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)
size = image.shape
width = size[0]
height = size[1]

center = (int(height/2),int(width/2))
M = cv2.getRotationMatrix2D(center, 45, 1)
imageRot = cv2.warpAffine(image, M, (height, width))
# Display image in a window
cv2.imshow("Output",image)
cv2.imshow("Rotate",imageRot)

# Wait until any key press (press any key to close the window)
cv2.waitKey()
# kill all the windows
cv2.destroyAllWindows()

```

### Expected Output:



## EDGE DETECTION:

- To detect the edges a function called **Canny** is used.

### Syntax:

```
edges = cv2.Canny(image, gradMin, gradMax)
```

- **gradMin** > Minimum intensity gradient
- **gradMax** > Maximum intensity gradient

### Example Code:

```
# Importing OpenCV Library
import cv2

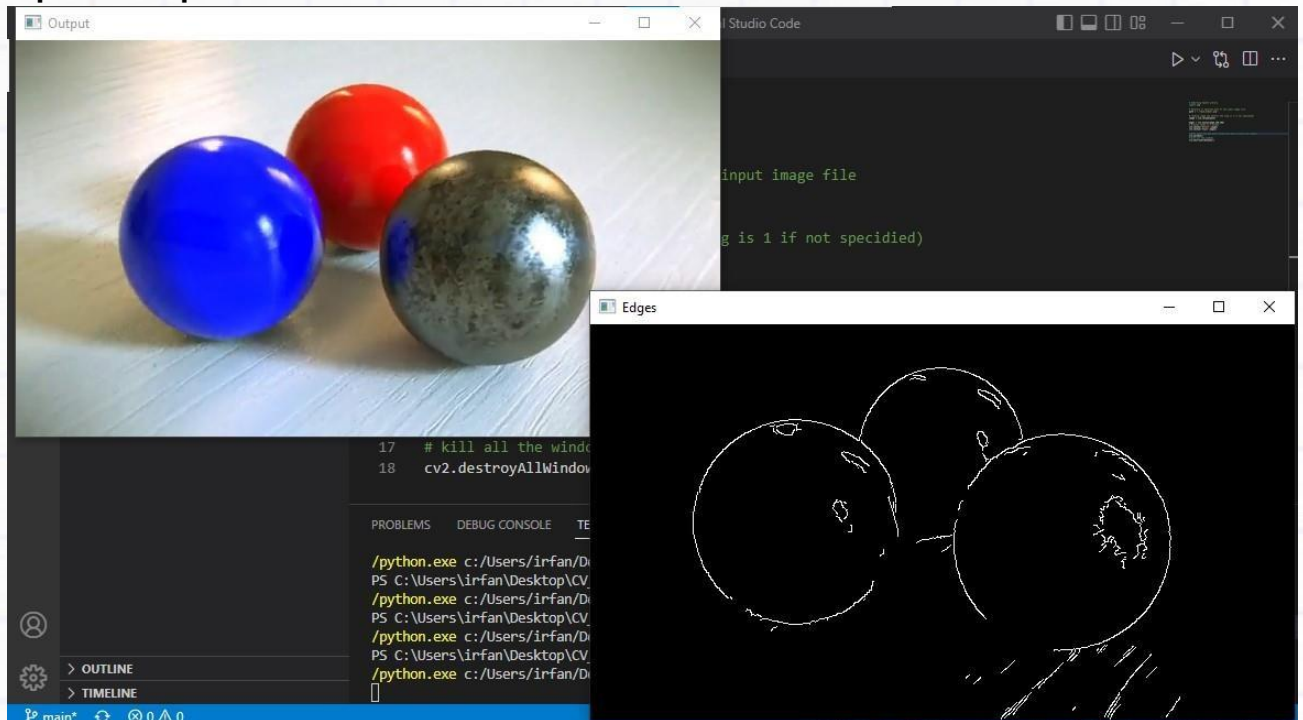
# Relative or absolute path of the input image file
path = "./task_6/ball.png"

# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)

edges = cv2.Canny(image,200,300)
# Display image in a window
cv2.imshow("Output",image)
cv2.imshow("Edges",edges)

# Wait until any key press (press any key to close the window)
cv2.waitKey()
# kill all the windows
cv2.destroyAllWindows()
```

## Expected Output:



## FINDING AND DRAWING CONTOURS:

- In order to find contours, we need a binary image, like the output of a canny edge detection. Since we already have it from the previous example we will be using `findContours` function on it that returns a group of contour points.
- After finding the contour points we can draw them on the original image and display them.

### Syntax:

```
contours, hierarchy = cv2.findContours(edges, retrieval_mode, approximation_method)
```

- `edges` > Binary Image
- `retrieval_mode` > contour retrieval mode
- `approximation_method` > contour approximation method
- For more details refer: [Contours Hierarchy](#)

```
cv2.drawContours(image, contours, contour_index, color, thickness)
```

- `contour_index` > if -1, draws all the contours; individual index can be given to draw a selective set of contours

### Example Code:

```
# Importing OpenCV Library
import cv2

# Relative or absolute path of the input image file
path = "./task_6/ball.png"

# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)

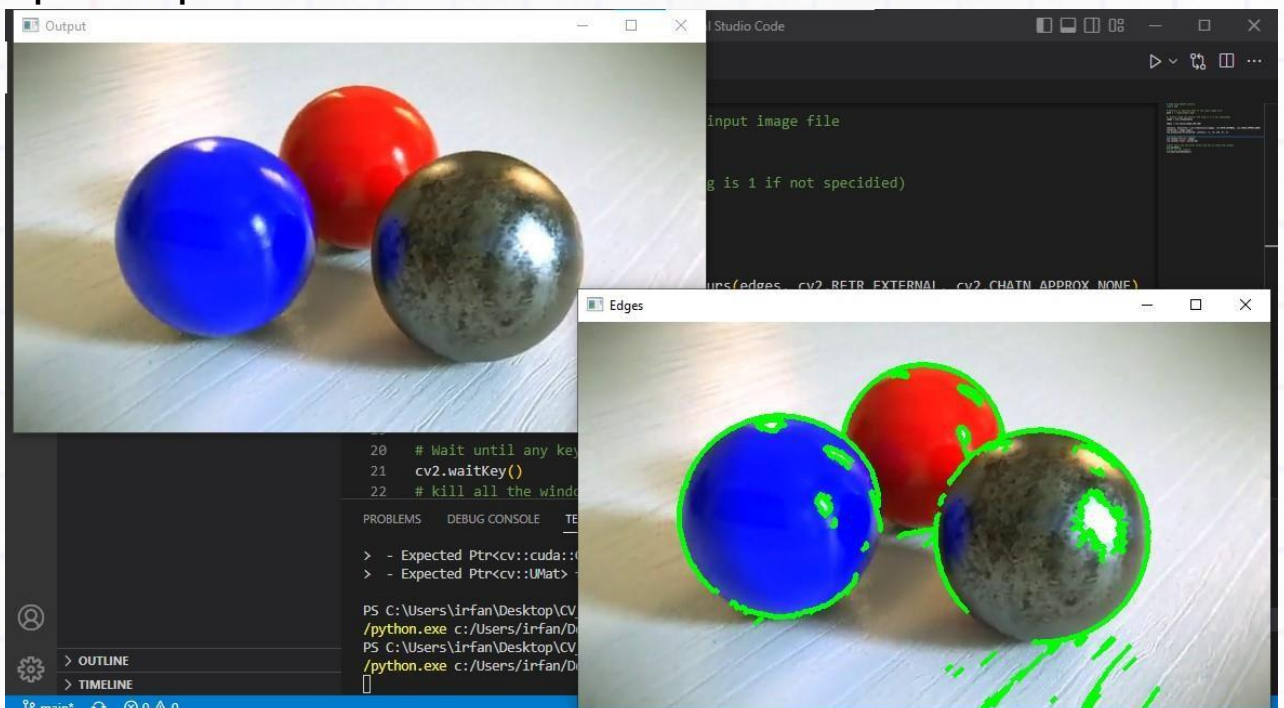
edges = cv2.Canny(image,200,300)

contours, hierarchy = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
contoured = image.copy()
cv2.drawContours(contoured, contours, -1, (0, 255, 0), 3)

# Display image in a window
cv2.imshow("Output",image)
cv2.imshow("Edges",contoured)

# Wait until any key press (press any key to close the window)
cv2.waitKey()
# kill all the windows
cv2.destroyAllWindows()
```

### Expected Output:



## SMOOTHENING:

- There are different methods of smoothing images in OpenCV. We will try out `medianBlur` and `GaussianBlur` in this task. For more details on smoothing refer: [Smoothing Images](#)

### Syntax:

```
cv2.medianBlur(image, kernel_size)
```

- `kernel_size` > size of the kernel for; here median of all the pixels under the kernel window is computed and the central pixel is replaced with this median value

```
GaussianBlur(image, kernel_size, sigma_x, sigma_y)
```

- `kernel_size` > instead of a box filter as in the case of a median blur, the width and height of the kernel need to be specified
- `sigma_x` > kernel standard deviation in the x direction
- `sigma_y` > kernel standard deviation in the y direction

### Example Code:

```
# Importing OpenCV Library
import cv2

# Relative or absolute path of the input image file
path = "./task_6/ball.png"

# reading image (by default the flag is 1 if not specified)
image = cv2.imread(path)

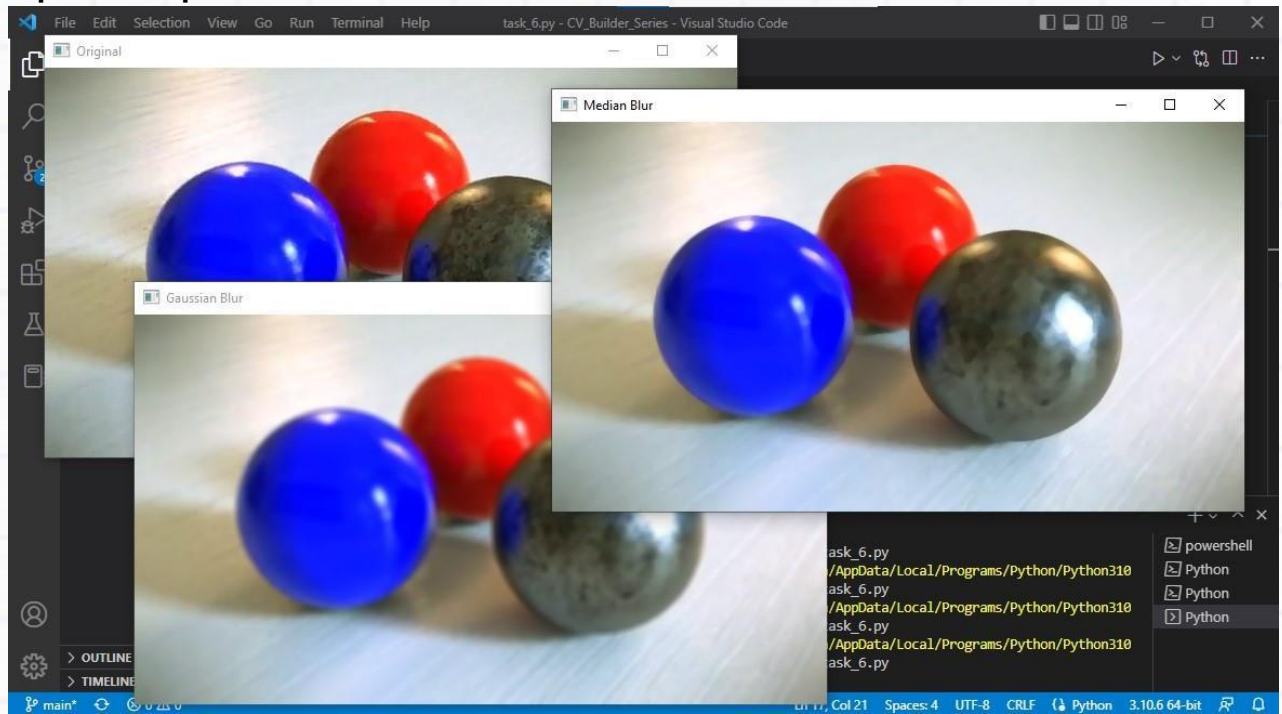
median = image.copy()
gaussian = image.copy()

median = cv2.medianBlur(median,7)
gaussian = cv2.GaussianBlur(gaussian, (7, 7), cv2.BORDER_DEFAULT)

# Display image in a window
cv2.imshow("Original",image)
cv2.imshow("Median Blur",median)
cv2.imshow("Gaussian Blur",gaussian)

# Wait until any key press (press any key to close the window)
cv2.waitKey()
# kill all the windows
cv2.destroyAllWindows()
```

## Expected Output:



## PERSPECTIVE WARPING:

- In order to perform perspective warping we need 4 points from the original image that we can stretch and convert to a top view. Therefore first, create a python file `task_6_2.py` that prints out the points whenever we left-click on the image. In this way, we can extract the 4 corner points for the warped image from the original image easily.

### Example Code:

```
# importing required libraries
import cv2
import numpy as np

p1 = (0,0) # top left corner point

# Mouse callback function
def mouseClicked(event, xPos, yPos, flags, param):

    global dp1

    # left click press event
    if event==cv2.EVENT_LBUTTONDOWN:
        p1 = (xPos,yPos)
        print(p1)

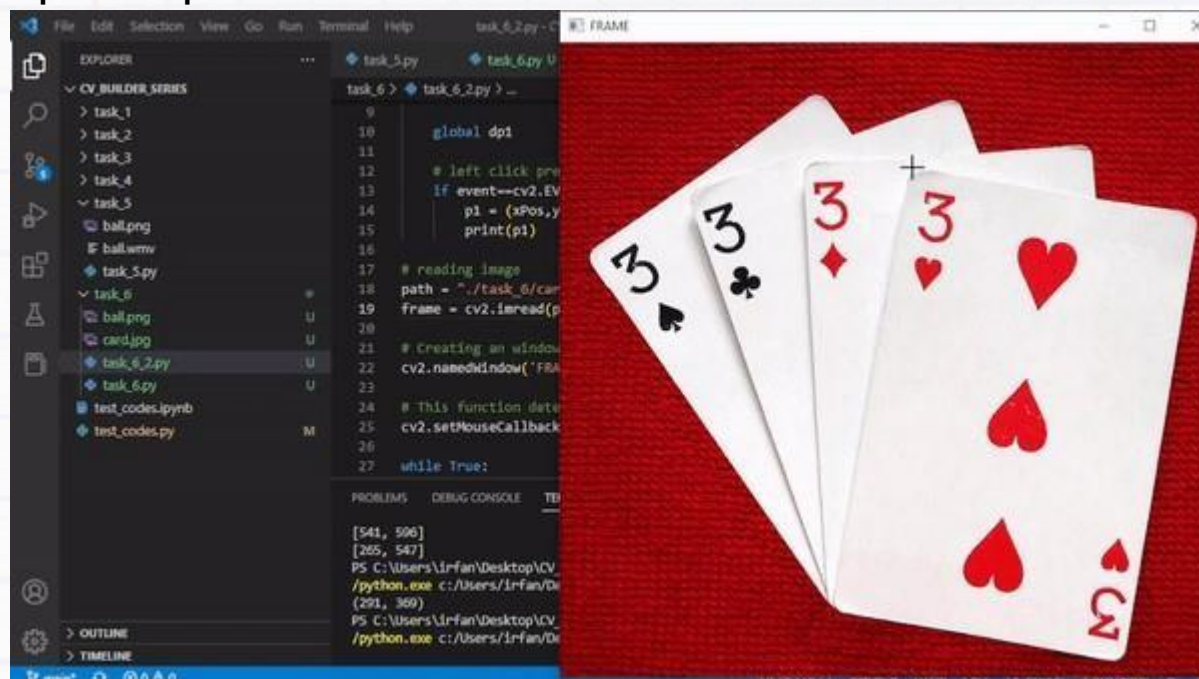
# reading image
path = "./task_6/card.jpg"
frame = cv2.imread(path)

# Creating an window to display image/frame
cv2.namedWindow('FRAME')

# This function detects every new events and triggers the "mouseClick"
function
cv2.setMouseCallback('FRAME',mouseClick)

while True:
    cv2.imshow('FRAME',frame)
    if cv2.waitKey(1) & 0xff == ord('q'): # to quit press 'q'
        break
cv2.destroyAllWindows()
```

## Expected Output:



- Now to warp this image, we need two sets of 4-points. One will be the source points where we can select points from the source image. And the other will be the destination points, these are the points where the source points will end up after transforming stretching, and warping the image. Both sets of points should be in the same order.
- This task is achieved by collecting the source points from a mouse click, and when we have 4 points. The `getPerspectiveTransform` function takes in the source and destination points and returns the transformation matrix.
- And `warpPerspective` warps the image according to the transformation matrix.

### Syntax:

```
matrix = cv2.getPerspectiveTransform(source_points,destination_points)
```

```
warped = cv2.warpPerspective(image,matrix,output_size)
```

### Example Code:

```
# importing required libraries
import cv2
import numpy as np

p1 = (0,0) # top left corner point
pts = []

# Mouse callback function
def mouseClicked(event,xPos,yPos,flags,param):

    global dp1

    # left click press event
    if event==cv2.EVENT_LBUTTONDOWN:
        p1 = (xPos,yPos)
        p1 = [p1[0],p1[1]]
        print(p1)
        pts.append(p1) # collecting corner points

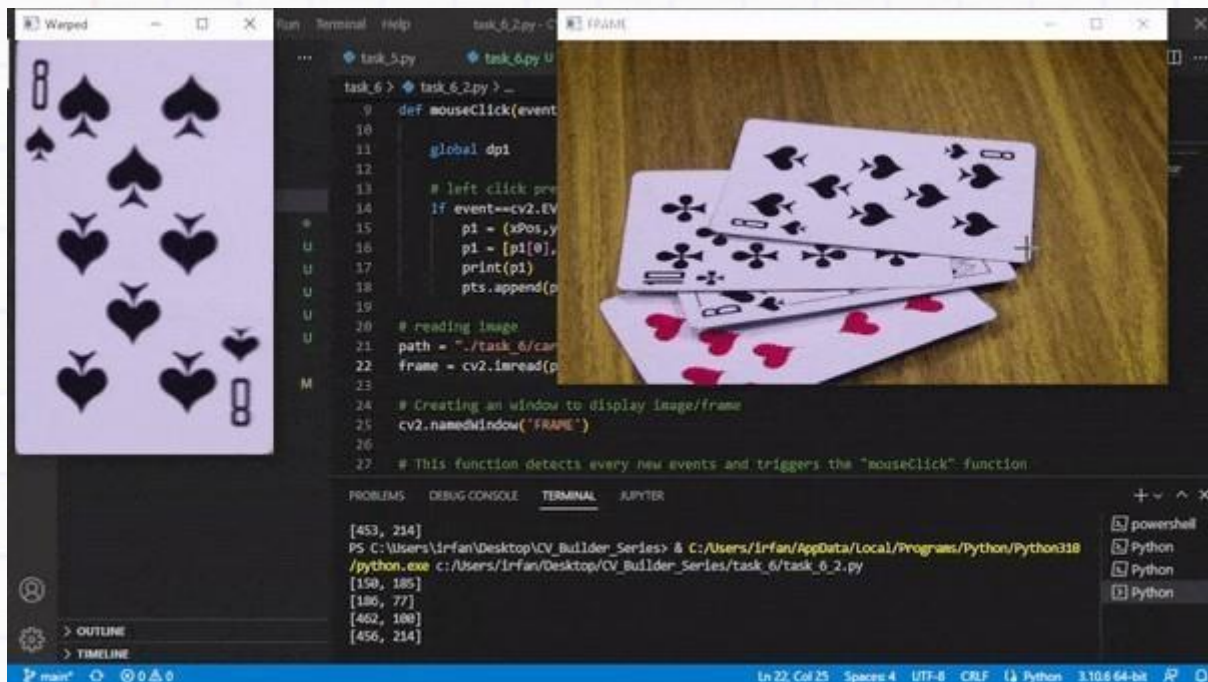
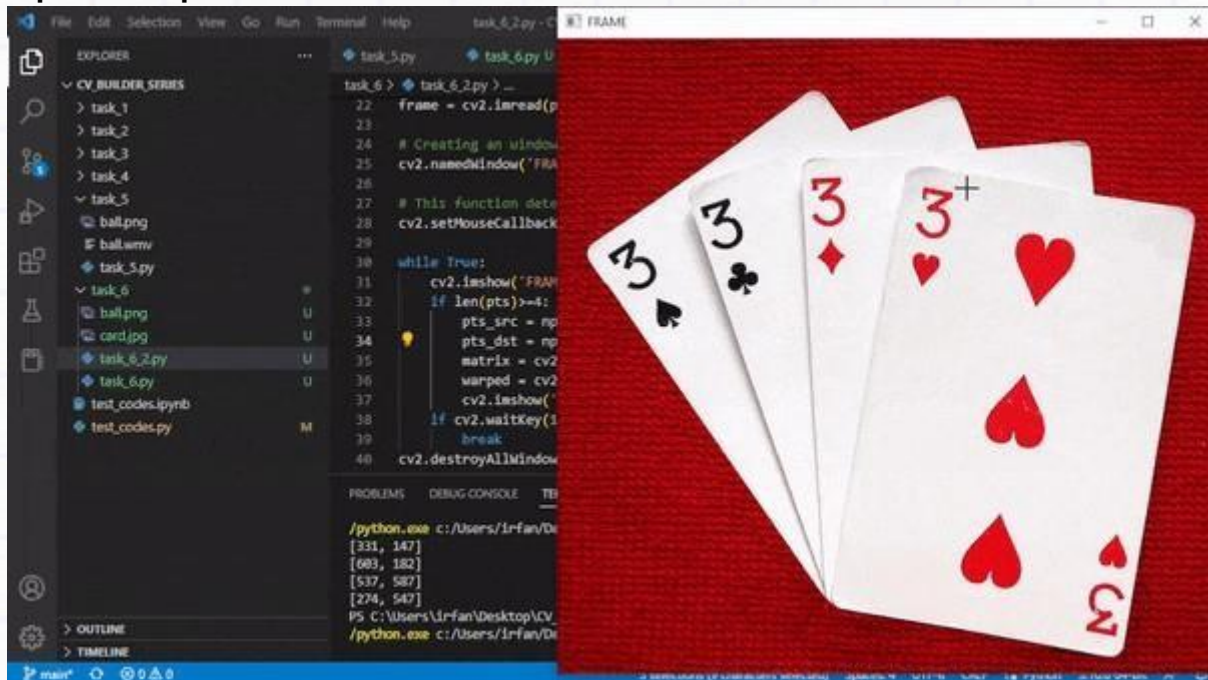
# reading image
path = "./task_6/card2.jpg"
frame = cv2.imread(path)

# Creating an window to display image/frame
cv2.namedWindow('FRAME')

# This function detects every new events and triggers the "mouseClick"
function
cv2.setMouseCallback('FRAME',mouseClick)

while True:
    cv2.imshow('FRAME',frame)
    if len(pts)>=4:
        pts_src = np.float32(pts[:4])
        pts_dst = np.float32([[0,0],[200,0],[200,400],[0,400]])
        matrix = cv2.getPerspectiveTransform(pts_src,pts_dst)
        warped = cv2.warpPerspective(frame,matrix,(200,400))
        cv2.imshow('Warped',warped)
        if cv2.waitKey(1) & 0xff == ord('q'): # to quit press 'q'
            break
    cv2.destroyAllWindows()
```

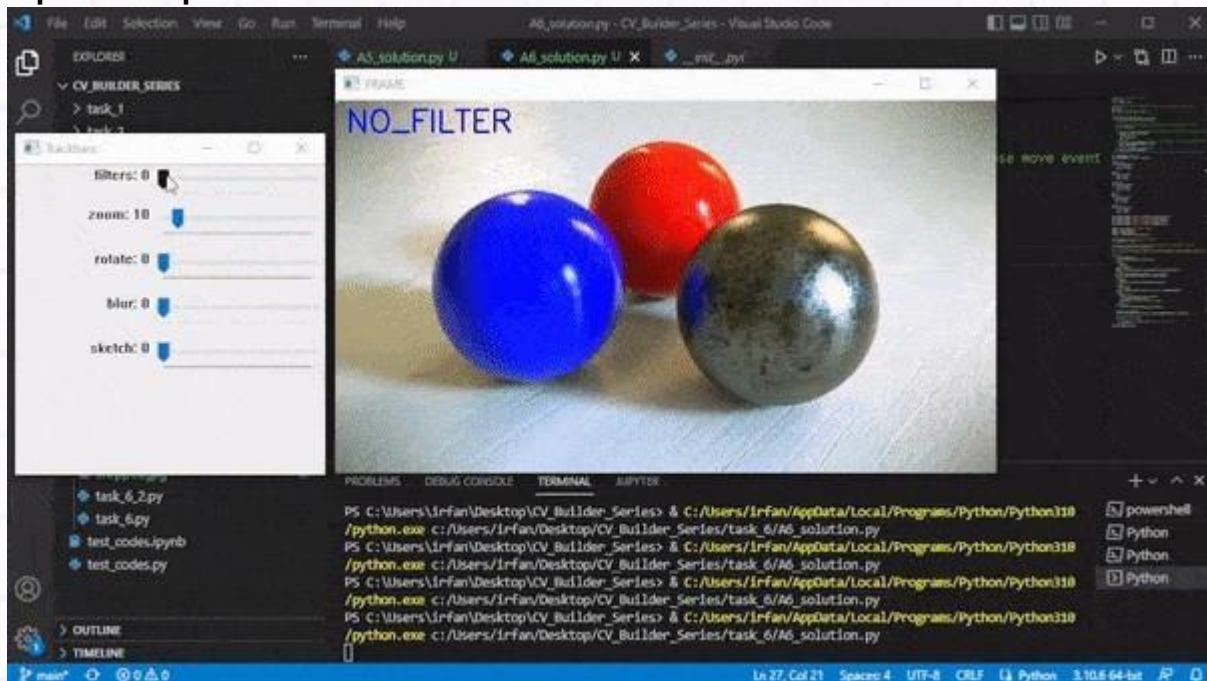
## Expected Output:



## WARMUP\_EXERCISE\_6:

- Create a very simple photo editor using track bars and tools covered in this section to achieve the following:
  - Filters, zooming, rotating, blurring, sketching effect (edge detection), and finally cropping and saving the cropped part of the image.
  - (Optional) For filter refer: [ColorMaps in OpenCV](#)

### Expected Output:



## 10. Face Detection and Recognition:

**Objective:** In this section you will learn how to detect a face, and recognize it. You will be using a template face which will be encoded and compared with the unknown faces in the camera or video feed.

### FACE DETECTION:

- For this task you will be using the `FaceDetection()` function of Mediapipe. It is quick and easy to implement.
- Before starting install Mediapipe using the command `pip install mediapipe` in your terminal. This should install latest version of Mediapipe.
- Make a file `task_7.py` inside a folder `task_7`. Create a face detector object using the function `mp.solutions.face_detection.FaceDetection()`
- You also need to get the video feed width and height using `cam.get()` function. Because after processing the frame the detector returns the location of face (`x_min`, `width` and `y_min`, `height`) which are normalized to  $0 \rightarrow 1$  by the video width and height respectively.

#### Syntax:

```
faces = mp.solutions.face_detection.FaceDetection()
```

```
faceResults = faces.process(frameRGB)
```

- `frameRGB` > You have to convert the frame to RGB before processing for face detection.
- `faceResults` > Returns a list of location of faces. (`x_min`, `width` and `y_min`, `height`)
  - `x_min`, `y_min` > top left corner of face bounding box.
  - `width`, `height` > width and height of bounding box from top left corner.

### Example Code:

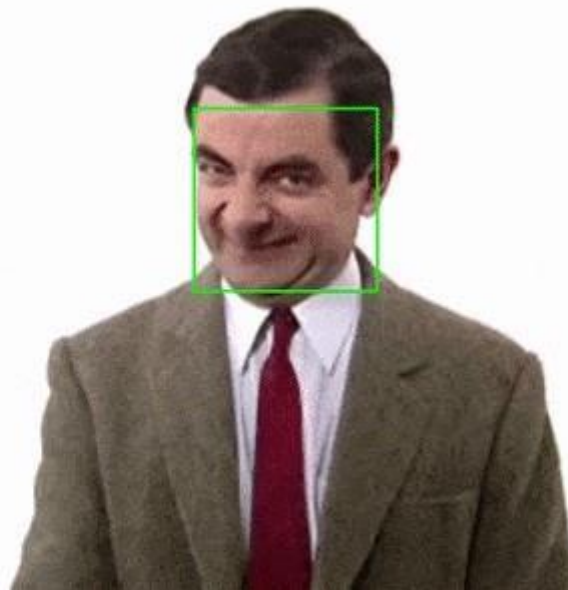
```
# import required libraries here
import cv2
import mediapipe as mp

# video capture object where 0 is the camera number for a usb camera (or
webcam)
# cam = cv2.VideoCapture(0)

# for video file, use this:
cam = cv2.VideoCapture("./task_7/mrBean.mp4")
# Frame width and height, will be useful later to find exact pixel locations
from normalized locations of faces
width = cam.get(cv2.CAP_PROP_FRAME_WIDTH)
height = cam.get(cv2.CAP_PROP_FRAME_HEIGHT)

# Creating a face detector object from Mediapipe solutions
faces = mp.solutions.face_detection.FaceDetection()
while True:
    _, frame = cam.read() # reading one frame from the camera object
    if _: # if frame received proceed
        frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) # Convert to
        RGB for processing
        faceResults = faces.process(frameRGB) # this returns list of
        location for all the faces in the current frame
        if faceResults.detections != None: # if detection is non
            empty or if atleast one face detected > proceed else
            skip
            for face in faceResults.detections: # iterate through
            each face locations
                bBox = face.location_data.relative_bounding_box #
                Collect bounding boxes for each face
                # splitting into variables and converting to
                integer for drawing rectangle around detected faces
                x,y,w,h =
                int(bBox.xmin*width),int(bBox.ymin*height),int(bBox
                .width*width),int(bBox. height*height)
                cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)
                cv2.imshow('Webcam', frame)
                # Waits for 1ms and check for the pressed key
                if cv2.waitKey(1) & 0xff == ord('q'): # press q to quit the camera
                    break
    cam.release() # close the camera
    cv2.destroyAllWindows() # Close all the active windows
```

## Expected Output:



## FACE RECOGNITION:

- Now let's build on the previous code to recognize Mr. Bean's face on a video. For this task you need to install `face_recognition` library.
- The `face_recognition` library has a dependency on `cmake`, hence you have to install `cmake` first.
- Install them with `pip install cmake` and `pip install face_recognition`
- Once you are ready with the library you can use the `face_encodings()` function to encode the template face in the beginning of the program.
- Within the loop after getting the bounding box, crop the face from the frame and encode that cropped face.
- You can compare the encoded template-face and the encoded unknown-face using the function `compare_faces()` and if it returns true, there is match. In this way you can recognize a person in a video or a photo.
- You can also do this for multiple people by template encoding faces of different people, or multiple faces of same person for better recognition of a single person.

### Syntax:

```
faceEncoding = fr.face_encodings(face)
```

```
match = fr.compare_faces(unknownEncoding, faceEncoding)
```

- Face > input template face.
- `unknownEncoding` > encoded object of template face.
- `faceEncoding` > encoded object of detected unknown face in current
- `frame.match` > `True` if there is a match, else `False`.

#### Example:

```
# import required libraries here
import cv2 import mediapipe as mp
import face_recognition as fr

# for video file, use:
cam = cv2.VideoCapture("./task_7/mrBean2.mp4")
# Frame width and height, will be useful later to find exact pixel locations
# from normalized locations of faces
width = cam.get(cv2.CAP_PROP_FRAME_WIDTH)
height = cam.get(cv2.CAP_PROP_FRAME_HEIGHT)
faces = mp.solutions.face_detection.FaceDetection()

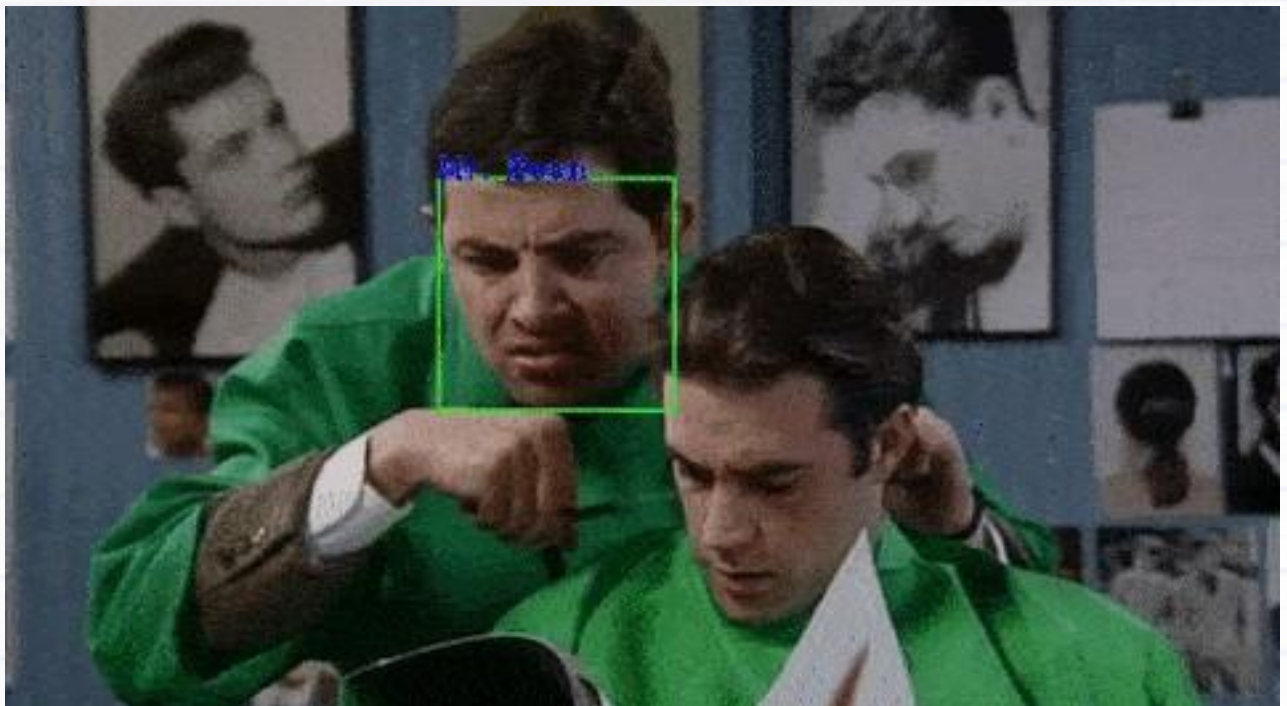
# loading template face image face =
fr.load_image_file("./task_7/mrBean.png") # load a face
# Encoding input face
faceEncoding = fr.face_encodings(face)[0] # since it returns list of encoded
faces, we are only providing one face in this example, and we are only
interested in the first face. Hence, [0]th face taken here.
while True:
    _, frame = cam.read() # reading one frame from the camera object
    if _: # if frame received proceed
        frameRGB = cv2.cvtColor(frame,cv2.COLOR_BGR2RGB) # convert to RGB
        for encoding
        faceResults = faces.process(frameRGB) # this returns list of
        location for all the faces in the current frame
        if faceResults.detections != None: # if detection is non empty or
        if atleast one face detected > proceed else skip
            for face in faceResults.detections: # iterate through each face
            locations
                bBox = face.location_data.relative_bounding_box # Collect
                bounding boxes for each face
                # splitting into variables and converting to integer for
                drawing rectangle around detected faces
                x,y,w,h =
                int(bBox.xmin*width),int(bBox.ymin*height),int(bBox.width*w
                idth),int(bBox.he ight*height)
                face = frameRGB[y:y+h,x:x+w] # getting cropped face
                if (face.shape[0]*face.shape[1]) > 0: # to filter out false
                detection and empty dimentioned crops
```

```

encoding = fr.face_encodings(face) # encoding unknown
face to compare with the template
match = fr.compare_faces(encoding,faceEncoding)
if True in match: # Draw rectangle and display name
if there is a match
    cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
    cv2.putText(frame,'Mr. Bean',
    (x,y),cv2.FONT_HERSHEY_COMPLEX,1,(255,0,0),2)
cv2.imshow('Webcam', frame)
cv2.waitKey(1) & 0xff == ord('q'): # press q to quit the camera
    break
cam.release() # close the camera
cv2.destroyAllWindows() # Close all the active windows

```

### Expected Output:



- Here you can see that sometimes it cannot recognize the face. This is because we are only using one face for template. More samples can be used for better recognition.

## WARMUP\_EXERCISE\_7:

- Build a basic attendance system using template images of your friends, it should be able to:
  - Detect faces from a video/camera feed and draw a box around the face.
  - Recognize the person by comparing the cropped face and template faces and display their name next to the bounding box.
  - Update the attendance, name, date and time in a text or csv file if there is a match.

### Expected Output:

```
Name, Attendance, Date, Time
Bean, Present, 2022-12-24, 08:12:55
Ben, Absent, -, -
Jacky, Absent, -, -
Robert, Present, 2022-12-24, 08:13:19
Tom, Present, 2022-12-24, 08:14:35
```

# Capstone Build Projects:

---

Follow the guided steps in order to finish this build task. Most of the OpenCV tools required to execute these tasks have already been covered in the tutorial above. Hence here we will not be providing the example codes. These build tasks will test your understanding of the concepts and tools from the tutorial.

## I. Posterization (Cartoonization) of portraits:

### PROBLEM STATEMENT:

- One of the more basic filters that one would find in apps like Photoshop Express or Google Snapseed is a Posterization or Cartoonification effect. It is an easy way to get a cool transformation of images, particularly portraits. The goal of this project is to be able to achieve this transformation.
- [Optional]: Once you have posterized the image, you can also try to transform the color space, and achieve visually striking images.
- Guided steps to achieve this are given below.

### GUIDED STEPS:

1. Import `cv2`, `matplotlib-pyplot`, and `numpy` which will be helpful in the later steps.

```
## Import necessary packages here
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

2. Read an image (selfie or your favorite celebrity) using `imread` function of OpenCV.

```
## Code for reading image from a location
```

3. Convert the input image from BGR format to RGB format using `cvtColor` function and then apply slight median blur using `medianBlur` function of OpenCV for smooth output. And display them using `pyplot` function of `matplotlib`



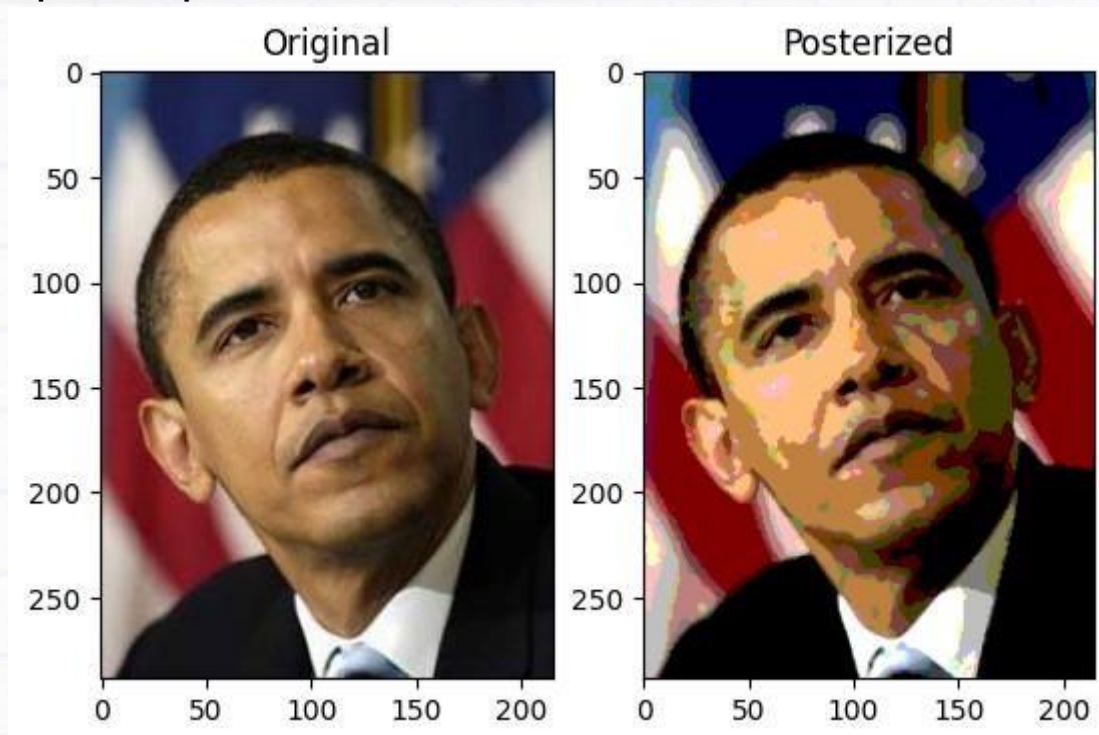
```
## Code for creating a Look Up Table for the given value of 'n'
```

5. Map input image pixel values according to the Look Up Table using `LUT` function of OpenCV.

```
## Code for mapping the input pixels with Look Up Table
```

6. Display the "Original and Posterized" image side-by-side using `pyplot` function of `matplotlib`.

**Expected Output for n = 5**



```
## Code for displaying "Original Vs Posterized Image"
```

## II. Finding Heart Rate (Pulse):

### PROBLEM STATEMENT:

- The beating of the heart can be observed visually by tiny variations in one's skin. This may not be visible to the untrained naked eye. One of the places where this would be more visible is on the tip of our finger. Though still very difficult to detect using the human eye, the variations at our fingertips are enough for camera sensors to pick up a pulse. This is the basis for apps such as Heart Rate Monitor on the Google Play Store.
- The objective of this project is to develop a program to measure your heart rate, and calculate your individual heart rate at different levels of activity:
  - RESTING ACTIVITY LEVELS: (When meditating)
  - FAT-BURN ACTIVITY LEVELS: (After walking around actively for a minute)
  - WORKOUT/CARDIO ACTIVITY LEVELS: (After jogging for a minute)
  - PEAK ACTIVITY LEVELS: (After climbing up and down stairs for over a minute)
- In order to be able to compute the heart rate, you will need to capture a video of your fingertip placed on the lens of your mobile camera after each of the above activity levels and record a 30-second clip. You can use this 30-second clip as a fixed input as you design your program. The final objective is to calculate your heart rate at each of the different activity levels.
- Guided steps to achieve this are given below.

### GUIDED STEPS:

1. Import the following packages:

- `cv2`
- `matplotlib.pyplot`
- `rfft, irfft, rfftfreq` from `scipy`
- `numpy`

```
# import necessary packages here
import cv2
from scipy.fft import rfft, rfftfreq, irfft # for fourier and inverse
fourier
import matplotlib.pyplot as plt # to display images within notebook
import numpy as np
```

2. Code a routine to read frames from the input pulse video and collect the green channel for each frame in a list.

```
# Code for collecting green channel of each frame in a list
```

3. Mean normalize the pixel values (Green Channel) collected by subtracting them from the mean of the whole signal. And plot it using `pyplot` function of `matplotlib`, to visualize the signal.

```
# Code for mean normalization here
```

4. Apply "Fast Fourier Transform" on the signal using the `rfft` function of `scipy`, and visualize it using `pyplot`.

```
# FFT code goes here
```

5. Filter out (Remove) frequencies outside the human pulse range and visualize:

- Lowest ever recorded: 27bpm > 0.45bps
- Highest ever recorded: 480bpm > 8bps

```
# Code to remove frequencies outside the desired range.
```

6. Find the frequency with the highest amplitude and multiply it by the time duration of the video. \$Beats

Per Minute (pulse) = Beats Per Second \* Video Duration (seconds)\$

```
# Code for extracting the pulse rate
```

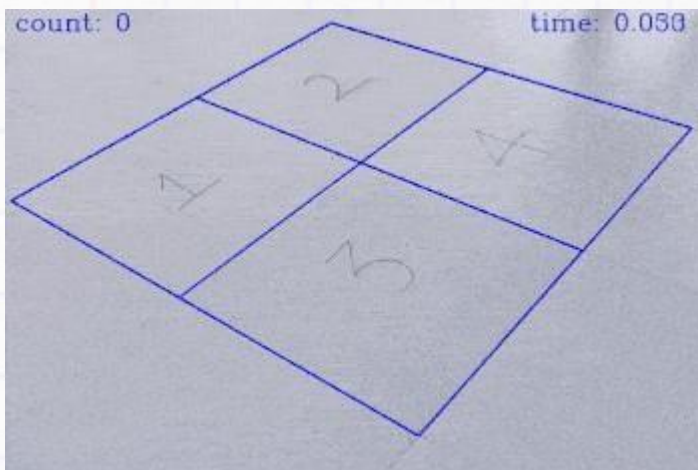
### III. Bounce Count:

#### PROBLEM STATEMENT:

- A video has been provided for this task and can be downloaded from here ([bounce](#)) in which a ball is dropped randomly on a 2x2 grid.
- Your task is to track the ball and detect when it touches the ground and, to find out when and in which region it bounced and also the total number of times it bounced from the ground.
- The program should draw the boundaries around the 4 quadrants, detect and track the ball, and print out the information as given below.

```
bounce_number, time_of_bounce, quadrant_of_bounce, frame_number
1, 0.718, 3, 49
2, 1.489, out_of_bound, 101
3, 2.560, 2, 180
4, 5.295, 2, 351
5, 6.118, 1, 401
```

- And the resulting video could of the format given below:

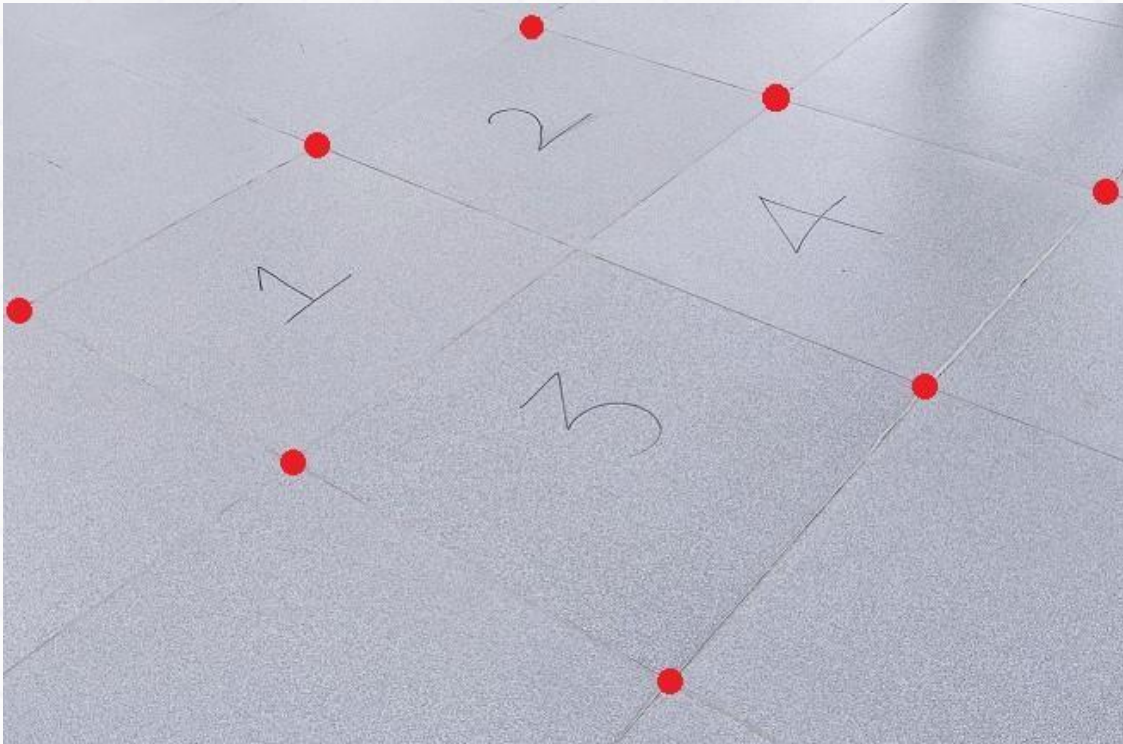


- Guided steps to achieve this are given below. (here you will be creating 2 helper programs and the main program. Hence it is good to work with python scripts unlike the previous 2 build tasks where you can do it with both Jupyter Notebook as well as Python scripts)

#### GUIDED STEPS:

1. First, take a screenshot of one of the frames from the video to generate the point coordinates from drawing the boundaries.

2. Write a program `point_coordinates.py` that uses a mouse click event and prints out the coordinates of the pixel where you click. And using this program extracts the points given below. (We did a similar program for warping in section 9, you can refer to that)



`point_coordinates.py`

```
# code for generating the point coordinates by mouse click
```

3. Write another program `mask.py` (similar to what we did for masking the ball in section 8) that prints out the lower bound and upper bound for the color of interest (green/yellowish ball in this case).

`mask.py`

```
# code for masking the ball and getting the lower and upper bound HSV range
```

4. Now you are ready for the main program. You can name it as `bounce.py`. Here importing `cv2` and `numpy` will be useful.
5. Create a video capture object by giving in the video file path, and get video properties like width, height, and fps using `get` function.
6. Get perspective transform matrix using `getPerspectiveTransform` method by passing in the 4 corners of the 4x4 grid collected from `point_coordinates.py`

7. Define a function that detects the ball using HSV masking and returns the contour points.
8. Define another function that returns the region of the ball when the ball hits the ground (it will be useful to use the warped frames/birds' eye view of the video to detect the region).
9. Now getting inside the loop, read the video frame by frame and draw lines around the 4x4 grid by using the points collected from `point_coordinates.py`
10. Get the region and current position of the ball. And then write a logic where you track the ball and find the exact moment it touches the ground (to make this problem less complex we recorded the video in such a way that you can look at the direction of the ball and detect when it touches the ground. The direction along the y coordinate flips the moment the ball bounces at the ground and returns)
11. If a bounce is detected, display the bounce count, region, and timestamp on the video. And also write it to a csv file for further analysis if required.

`bounce.py`

```
# import cv2 and numpy

# create video capture object

# get width, height and fps

# function for ball detection

# function for region detection

#-----LOOP-----

# 1. read frame

# 2. get ball position from ball detecting function

# 3. get current ball region from region detection function

# 4. Logic to detect bounce
    # if bounce detected display bounce count, region of bounce, time of
    bounce on video
    # also write it to a csv file for further analysis if required

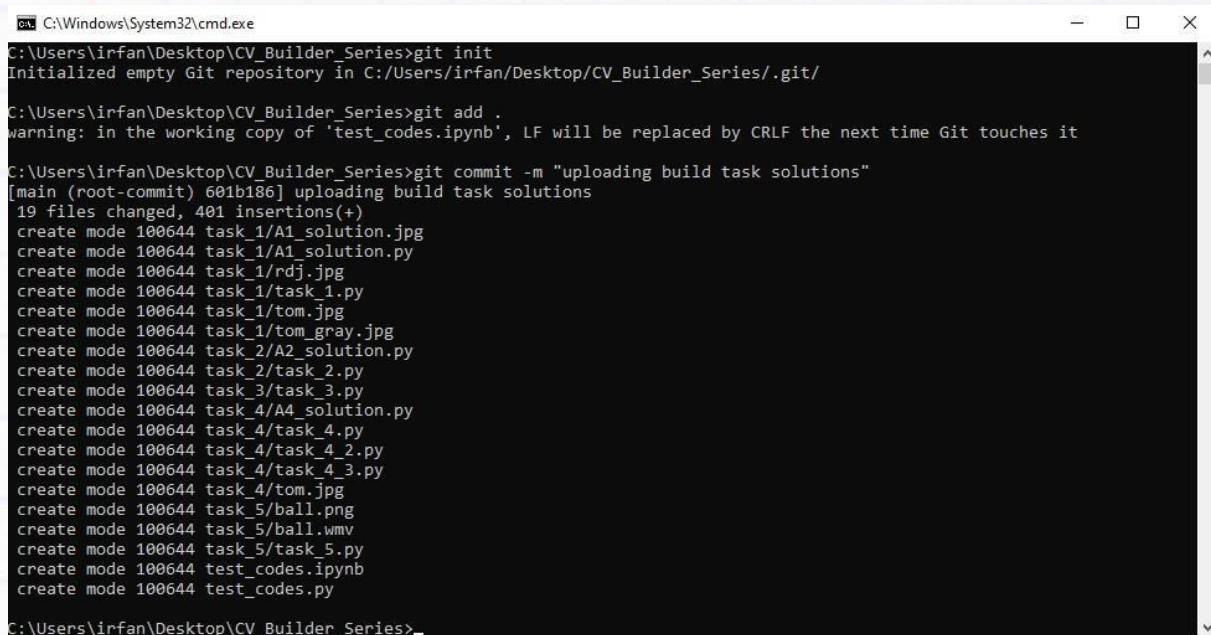
# 5. display timer on video

#----- END LOOP -----
```

# Uploading projects to GitHub:

**Objective: To upload your build task projects on your personal GitHub. Follow the step-by-step instruction given below to finish the task.**

- Before starting make sure that you have a GitHub account and GIT installed on your computer already.
- Open your working folder which is "CV\_Builder\_Series".
- Open the Command prompt (for windows users) or terminal (for ubuntu/linux users) in your working folder and run the command `git init` which will initialize an empty Git repository in your working folder.
- Then run `git add .` to add all your changes into the staging area for commit.
- Run `git commit -m "uploading build task solutions"` to commit your changes to your local repository.

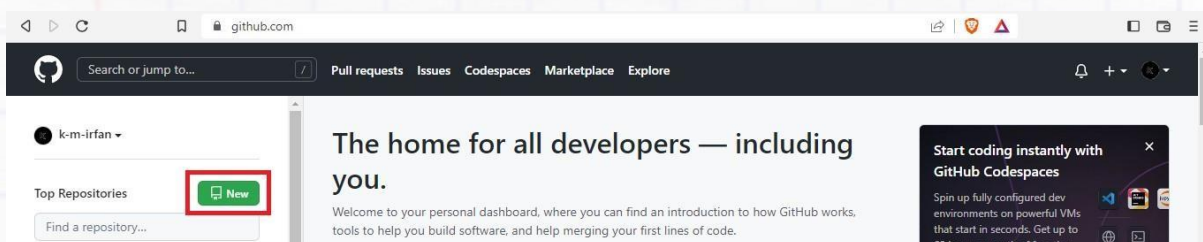


```
C:\Windows\System32\cmd.exe
C:\Users\irfan\Desktop\CV_Builder_Series>git init
Initialized empty Git repository in C:/Users/irfan/Desktop/CV_Builder_Series/.git/

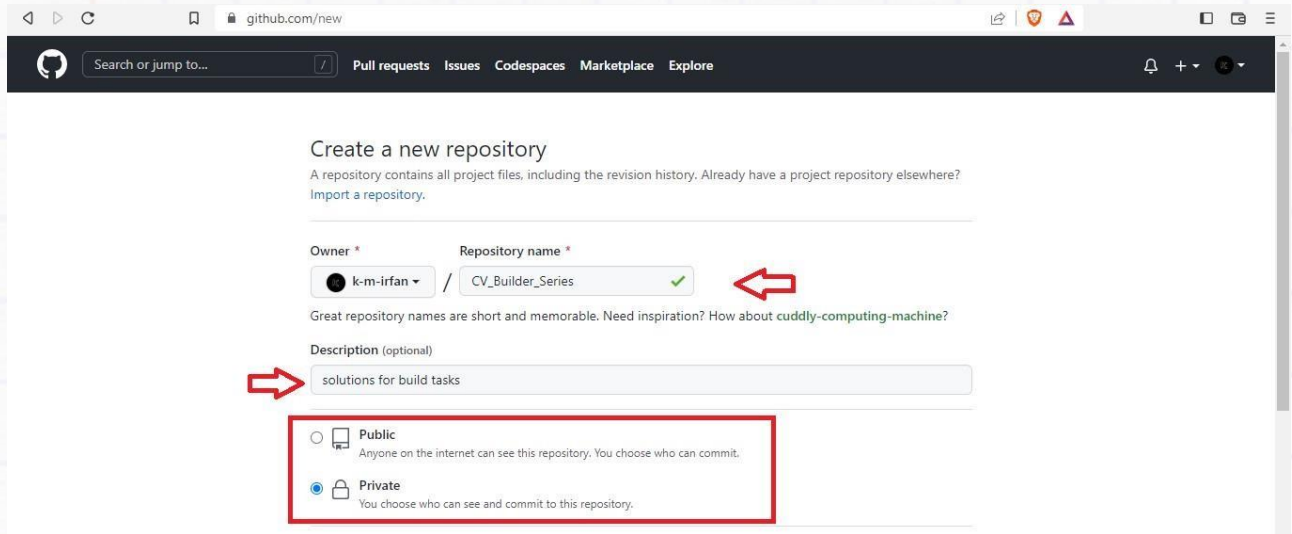
C:\Users\irfan\Desktop\CV_Builder_Series>git add .
warning: in the working copy of 'test_codes.ipynb', LF will be replaced by CRLF the next time Git touches it

C:\Users\irfan\Desktop\CV_Builder_Series>git commit -m "uploading build task solutions"
[main (root-commit) 601b186] uploading build task solutions
19 files changed, 401 insertions(+)
create mode 100644 task_1/A1_solution.jpg
create mode 100644 task_1/A1_solution.py
create mode 100644 task_1/rdj.jpg
create mode 100644 task_1/task_1.py
create mode 100644 task_1/tom.jpg
create mode 100644 task_1/tom_gray.jpg
create mode 100644 task_2/A2_solution.py
create mode 100644 task_2/task_2.py
create mode 100644 task_3/task_3.py
create mode 100644 task_4/A4_solution.py
create mode 100644 task_4/task_4.py
create mode 100644 task_4/task_4_2.py
create mode 100644 task_4/task_4_3.py
create mode 100644 task_4/tom.jpg
create mode 100644 task_5/ball.png
create mode 100644 task_5/ball.wmv
create mode 100644 task_5/task_5.py
create mode 100644 test_codes.ipynb
create mode 100644 test_codes.py
C:\Users\irfan\Desktop\CV_Builder_Series>
```

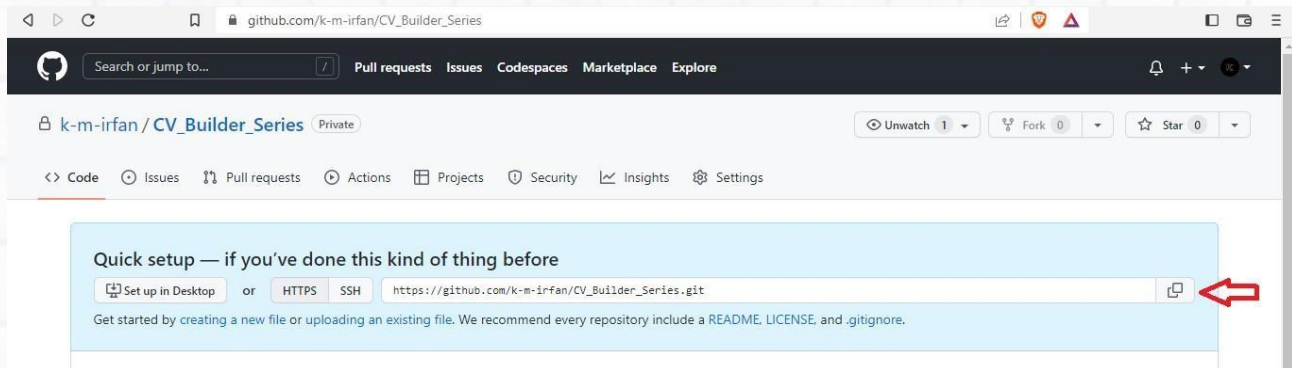
- Now before moving forward we need to create a repo with the same name in our GitHub.
- Open a GitHub account and add a new repository



- Give the same name "CV\_Builder\_Series" to the repository. Add a description and make it private or public as per your choice. We will not be adding readme for now.

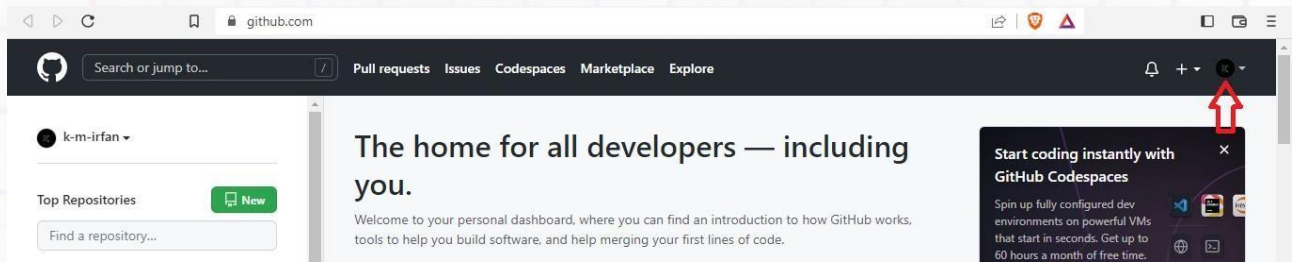


- Scroll down and click on "create repository".
- Copy the repo link:

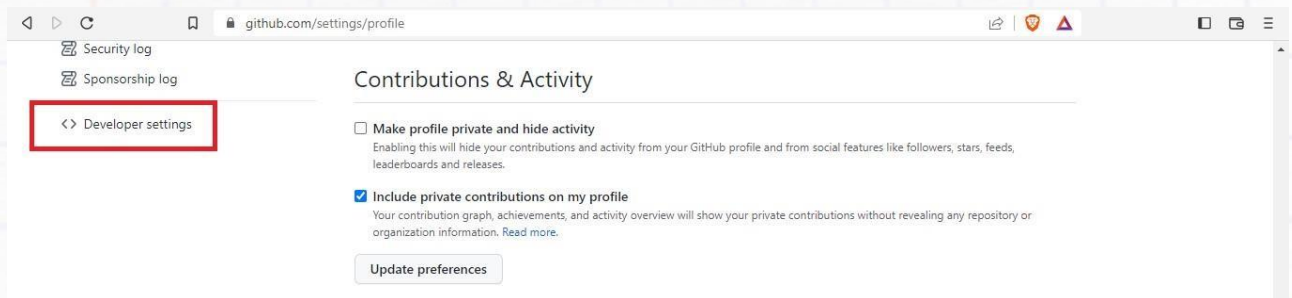


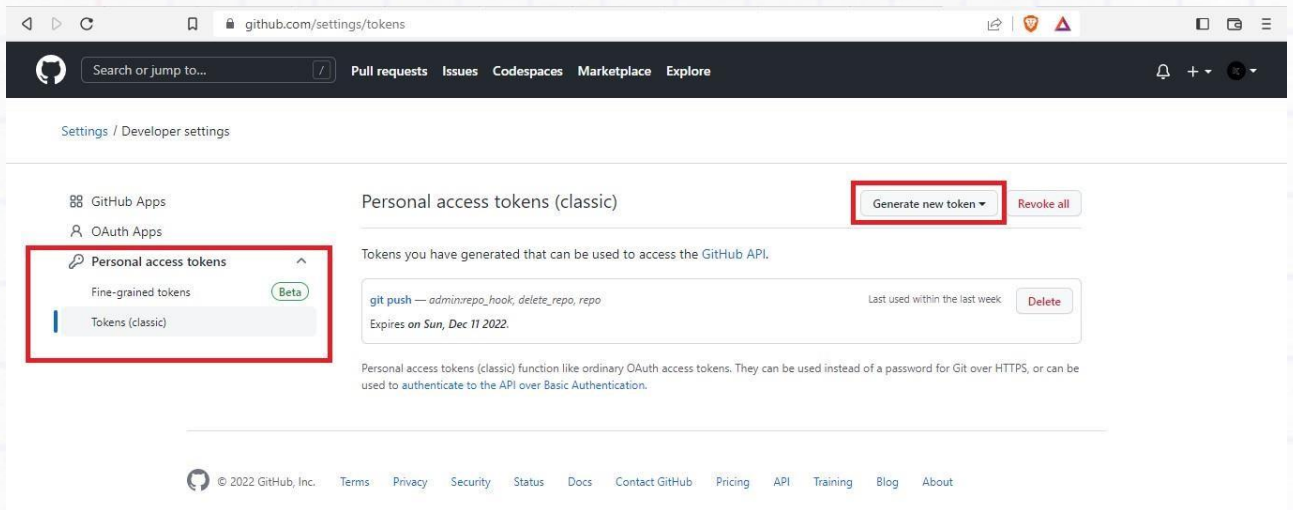
- Now come back to the terminal/command prompt and run this command: `git remote add origin https://github.com/**your_username**/your_project_name.git`. This will link the remote and the local repository. Don't forget to add your username in the above link.
- Create the main branch using the command: `git branch -M main`
- Now you are ready to push your project files to GitHub. Run `git push -u origin main` to push the commits to GitHub.
- If you are doing it for the first time GIT may ask for your GitHub ID and password. In place of a password you have to provide a token that can be generated by following the steps below:

- Click on your photo on GitHub and open settings:

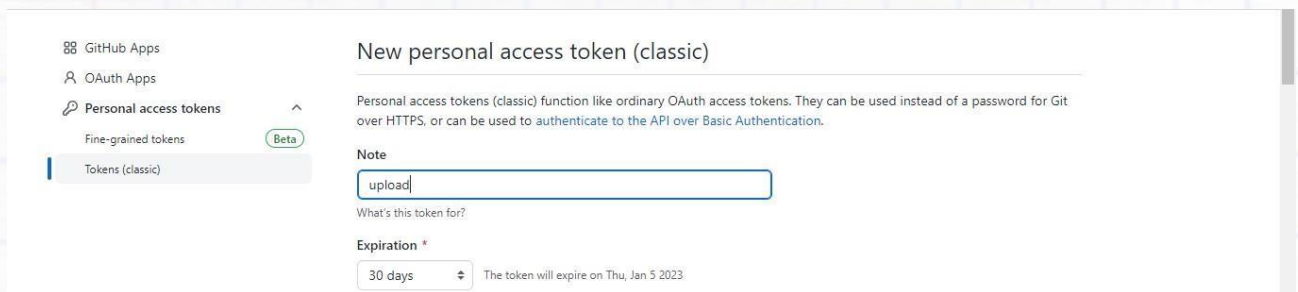


- Scroll down and click on Developer settings on the left panel to generate a token (classic):

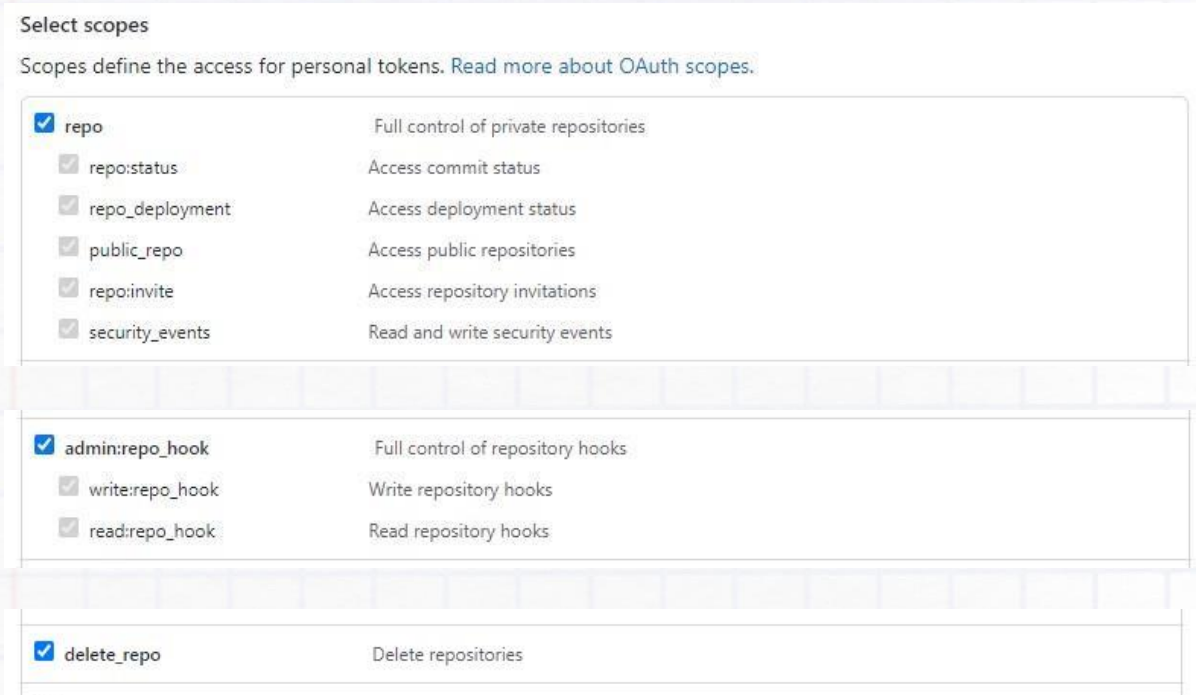




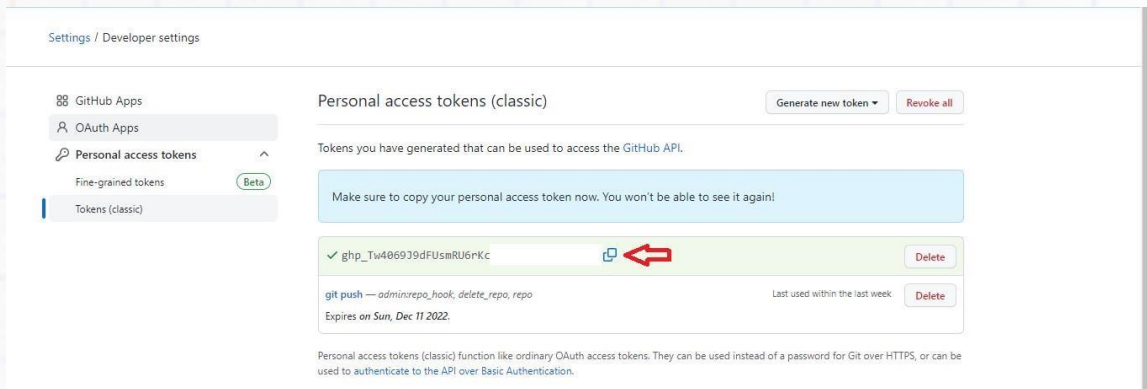
- Give a note. (like for what purpose you will be using this token).



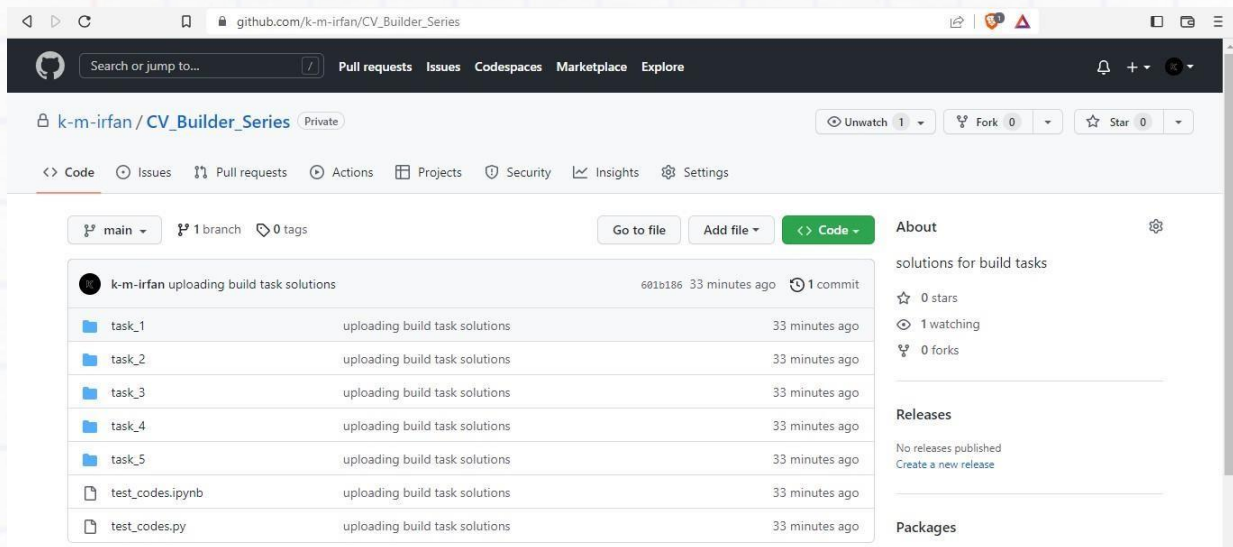
- Select scopes and click on "generate token" at the bottom of the page.



Copy the token and save it somewhere, you will not be able to see this again.



- Finally, if the **push** was successful, reload or navigate to your repo, and you can see all your files have been uploaded to GitHub.



## WHAT NEXT?

- You can read more and play around with OpenCV and Mediapipe to build cool projects: You can refer these resources below to explore and experiment more in this area.
  - Checkout this website where you can find bunch of cool projects and example codes to play with: [LearnOpenCV](#)
  - Play around and experiment with the different solutions that Mediapipe offers, like detecting face, hands, pose landmarks, pose and hair segmentation etc. You can find example codes for most of the solutions they provide: [GitHub-Mediapipe](#)
  - Also for a very detailed YouTube tutorial by Paul McWhorter, checkout this playlist: [AI For Everyone](#)

End of Build Series!