

Secure Virtual Key Entry

Aim : Develop a mechanism to secure the virtual key entry.

Methodology : 1. Two-factor authentication (2FA)

1. Facial recognition
2. OTP verification

Two-factor authentication (2FA)

Two-factor authentication (2FA), sometimes referred to as two-step verification or dual-factor authentication, is a security process in which users provide two different authentication factors to verify themselves.

2FA is implemented to better protect both a user's credentials and the resources the user can access. Two-factor authentication provides a higher level of security than authentication methods that depend on single-factor authentication (SFA), in which the user provides only one factor -- typically, a password or passcode. Two-factor authentication methods rely on a user providing a password as the first factor and a second, different factor -- usually either a security token or a biometric factor, such as a fingerprint or facial scan.

Two-factor authentication adds an additional layer of security to the authentication process by making it harder for attackers to gain access to a person's devices or online accounts because, even if the victim's password is hacked, a password alone is not enough to pass the authentication check.

Two-factor authentication has long been used to control access to sensitive systems and data. Online service providers are increasingly using 2FA to protect their users' credentials from being used by hackers who stole a password database or used phishing campaigns to obtain user passwords.

The Two-factor authentication we are going to use here for Securing the Virtual Key Entry are

1. Facial recognition
2. OTP verification

Facial recognition

Facial recognition is a way of identifying or confirming an individual's identity using their face. Facial recognition systems can be used to identify people in photos, videos, or in real-time.

Recognize and manipulate faces from Python or from the command line with the world's simplest face recognition library. Built using dlib's state-of-the-art face recognition built with deep learning. The model has an accuracy of 99.38% on the Labeled Faces in the Wild benchmark.

This also provides a simple face_recognition command line tool that lets you do face recognition on a folder of images from the command line! Many people are familiar with face recognition technology through the FaceID used to unlock iPhones (however, this is only one application of face recognition). Typically, facial recognition does not rely on a massive database of photos to determine an individual's identity — it simply identifies and recognizes one person as the sole owner of the device, while limiting access to others.

Beyond unlocking phones, facial recognition works by matching the faces of people walking past special cameras, to images of people on a watch list. The watch lists can contain pictures of anyone, including people who are not suspected of any wrongdoing, and the images can come from anywhere — even from our social media accounts. Facial technology systems can vary, but in general, they tend to operate as follows:

Step 1: Face detection

The camera detects and locates the image of a face, either alone or in a crowd. The image may show the person looking straight ahead or in profile.

Step 2: Face analysis

Next, an image of the face is captured and analyzed. Most facial recognition technology relies on 2D rather than 3D images because it can more conveniently match a 2D image with public photos or those in a database. The software reads the geometry of your face. Key factors include the distance between your eyes, the depth of your eye sockets, the distance from forehead to chin, the shape of your cheekbones, and the contour of the lips, ears, and chin. The aim is to identify the facial landmarks that are key to distinguishing your face.

Step 3: Converting the image to data

The face capture process transforms analog information (a face) into a set of digital information (data) based on the person's facial features. Your face's analysis is essentially turned into a mathematical formula. The numerical code is called a faceprint. In the same way that thumbprints are unique, each person has their own face

Step 4: Finding a match

- Find and recognize unknown faces in a photograph based on photographs of known people
- Compare faces by numeric face distance instead of only True/False matches
- Recognize faces in live video using your webcam - Simple / Slower Version (Requires OpenCV to be installed)
- Recognize faces in live video using your webcam - Faster Version (Requires OpenCV to be installed)
- Recognize faces in a video file and write out new video file (Requires OpenCV to be installed)
- Recognize faces on a Raspberry Pi w/ camera
- Run a web service to recognize faces via HTTP (Requires Flask to be installed)
- Recognize faces with a K-nearest neighbors classifier

One-Time Passcode Verification (OTP)

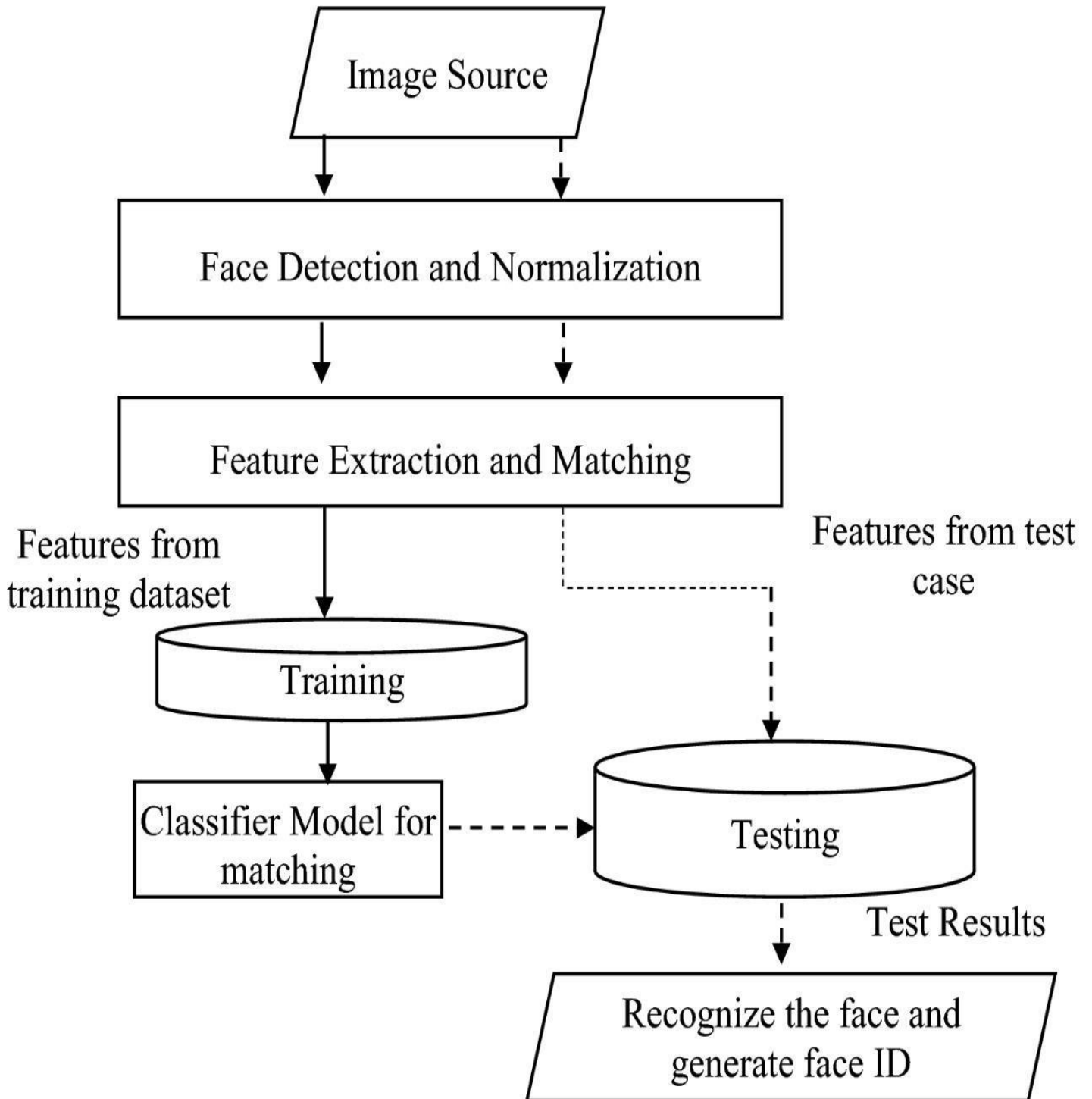
After Facial recognition OTP Verification verifies the person by sending OTP verification code during login and contact form submissions and It removes the possibility of users registering with fake Email Address by enabling OTP Verification.

We can easily create an application for the task of OTP verification using Python by following the steps mentioned below:

1. First, create a 6-digit random number
2. Then store the number in a variable
3. Then we need to write a program to send emails
4. When sending email, we need to use OTP as a message
5. Finally, we need to request two user inputs; first for the user's email and then for the OTP that the user has received.

So this is the complete process of creating an OTP verification application using Python. In the section below, I will take you through how to implement these steps using Python for the task of OTP verification.

Flowchart :



Program for Secure Virtual Key Entry

```
import random
import smtplib
import face_recognition as fr # To recognize faces
import numpy as np # To handle all lists/arrays
import cv2 # To capture webcam footage
import os # To handle all matters relating to folders, paths, image/file names, etc.

# Path to folder containing all known faces. The 'known' folder needs all known faces,
# where the image\file name is the name of the person to be recognized.
faces_path = "C:\Users\artam\PycharmProjects\face_reco\face"
n = ""
em = ''

# Function to get face names, as well as face encodings
def get_face_encodings():
    face_names = os.listdir(faces_path)
    face_encodings = []
    # For loop to retrieve all face encodings and store them in a list.
    # Below loop also gets the names of people and removes ".jpg", and stores
    # the names in a list
    for i, name in enumerate(face_names):
        face = fr.load_image_file(f"{faces_path}\\{name}")
        face_encodings.append(fr.face_encodings(face)[0])
        face_names[i] = name.split(".")[0] # To remove ".jpg" or any other image extension
    return face_encodings, face_names

# Retrieving face encodings and storing them in the face_encodings variable, along with the
names
face_encodings, face_names = get_face_encodings()
# Reference to webcam
video = cv2.VideoCapture(0)
#video.set(cv2.CAP_PROP_FPS, 420)
# Setting variable which will be used to scale size of image
scl = 2
# Continuously capturing webcam footage
while True:
    success, image = video.read()
    # Making current frame smaller so program runs faster
    resized_image = cv2.resize(image, (int(image.shape[1] / scl), int(image.shape[0] / scl)))
    # Converting current frame to RGB, since that's what the face recognition module uses
    rgb_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)
    # Retrieving face location coordinates and unknown encodings
    face_locations = fr.face_locations(rgb_image)
    unknown_encodings = fr.face_encodings(rgb_image, face_locations)
    # Iterating through each encoding, as well as the face's location
    for face_encoding, face_location in zip(unknown_encodings, face_locations):
        # Comparing known faces with unknown faces
        result = fr.compare_faces(face_encodings, face_encoding, 0.4)
        # Getting correct name if a match was found
        if True in result:
            name = face_names[result.index(True)]
            # Setting coordinates for face location
            top, right, bottom, left = face_location
            # Drawing rectangle around face
            cv2.rectangle(image, (left * scl, top * scl), (right * scl, bottom * scl), (0, 0,
255), 2)

            # Setting font, as well as displaying text of name
            font = cv2.FONT_HERSHEY_DUPLEX
```

```

cv2.putText(image, name, (left * scl, bottom * scl + 20), font, 0.8, (255, 255,
255), 1)

n = name
print(n)

cv2.imshow("frame", image)
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login("rahulmydur341@gmail.com", 'setvmwylpfgkjown')
otp = ''.join([str(random.randint(0, 9)) for i in range(6)])
msg = 'Hello, Your OTP is ' + str(otp)

if n == 'Rahul':
    em = 'rahulmydur@gmail.com'
    server.sendmail('rahulmydur341@gmail.com', em, msg)
    server.quit()
    break
    print("Hello rahul")
elif n == "pawan":
    em = '7pawancena@gmail.com'
    server.sendmail('rahulmydur341@gmail.com', em, msg)
    server.quit()
    break
    print("Hello pawan")
elif n == "manoj":
    em = 'manojmanugmail.com'
    server.sendmail('rahulmydur341@gmail.com', em, msg)
    server.quit()
    break
    print("Hello manoj")

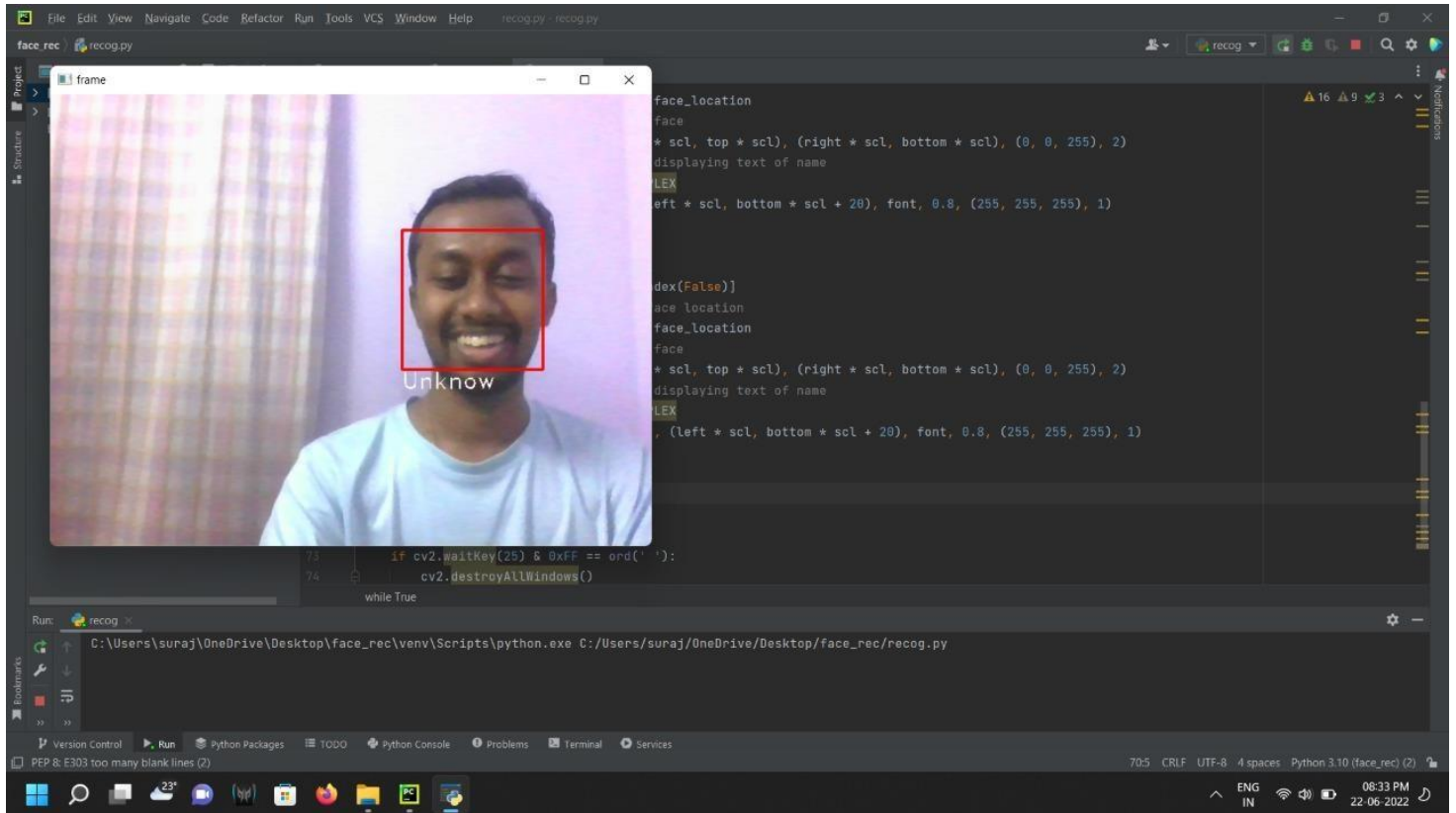
if cv2.waitKey(25) & 0xFF == ord(' '):
    break

otp_m = int(otp)
correct_otp = int(input("Enter the OTP : "))
if correct_otp == otp_m:
    print("Access granted")
else:
    print("Incorrect OTP ")

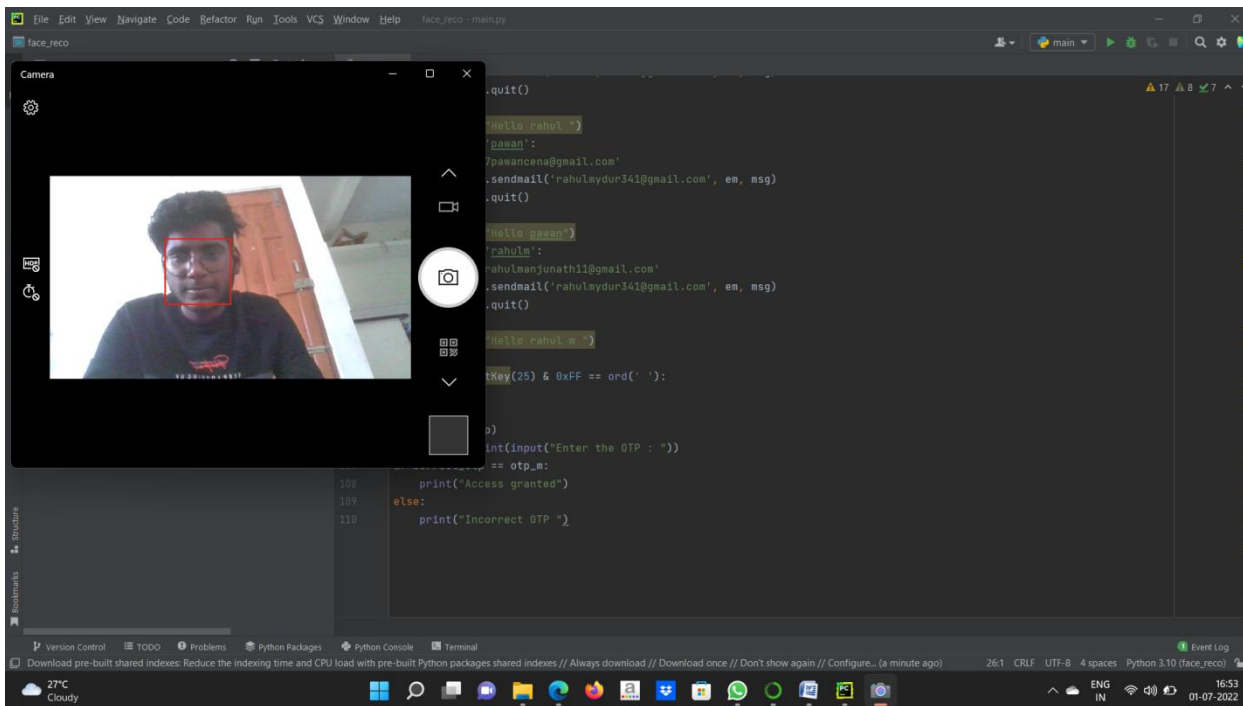
```

Results

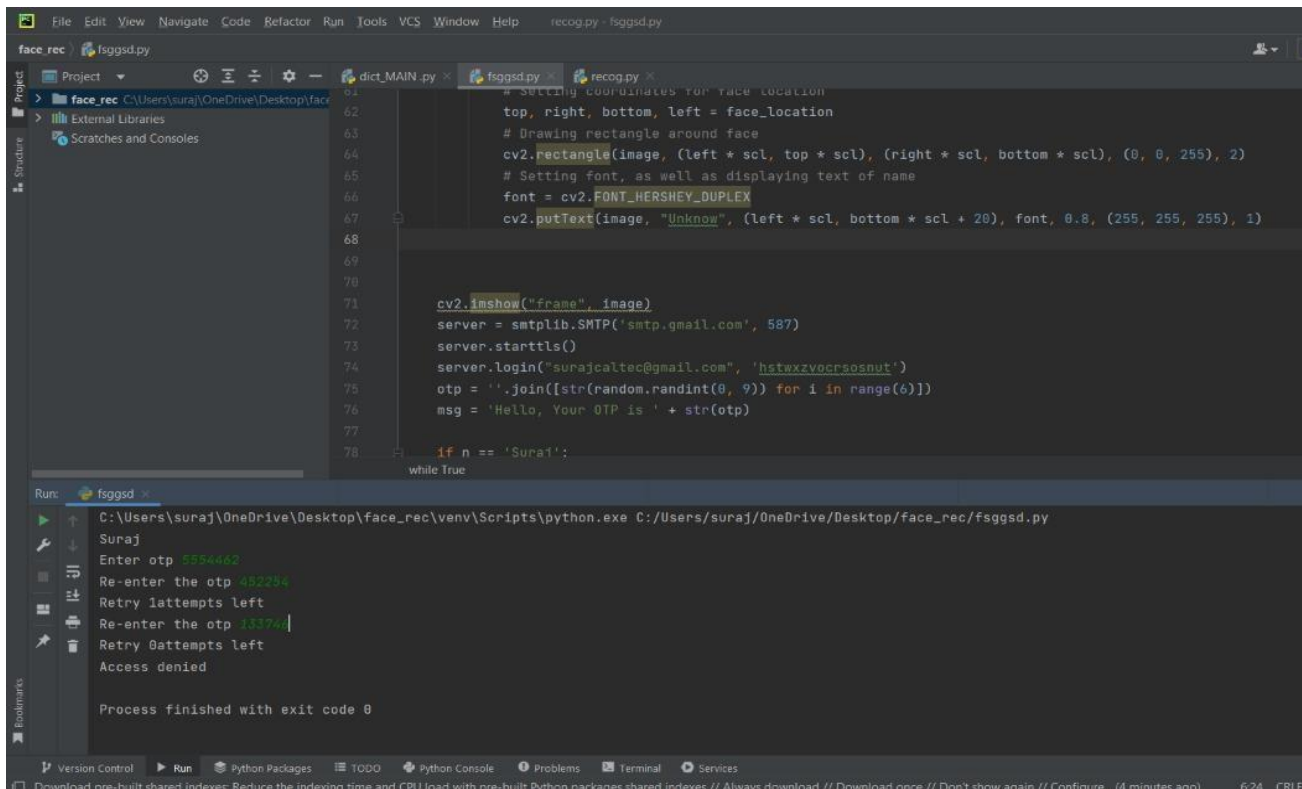
1. When unknown face is recognition :



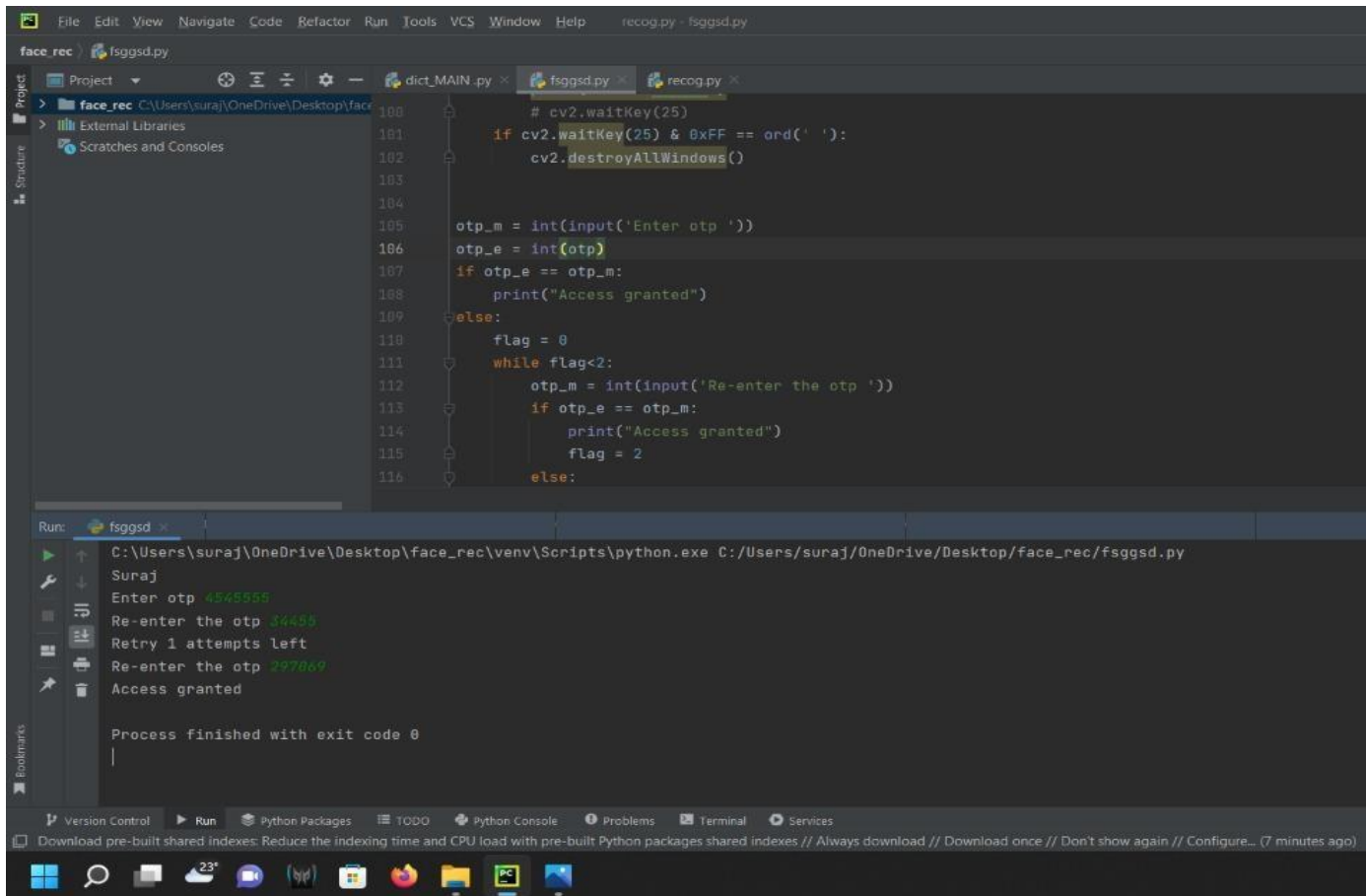
2. When known face is recognition :



3. When a invalid OTP entered :



3. When a valid OTP entered :



The screenshot displays a code editor with a Python script for OTP verification. The code prompts the user to enter an OTP, checks if it matches a stored value, and allows for a second attempt if the first one fails. The output window shows the user's input and the program's response.

```
100 # cv2.waitKey(25)
101 if cv2.waitKey(25) & 0xFF == ord(' '):
102     cv2.destroyAllWindows()
103
104
105 otp_m = int(input('Enter otp '))
106 otp_e = int(otp)
107 if otp_e == otp_m:
108     print("Access granted")
109 else:
110     flag = 0
111     while flag<2:
112         otp_m = int(input('Re-enter the otp '))
113         if otp_e == otp_m:
114             print("Access granted")
115             flag = 2
116         else:
```

Run: fsggsd x

C:\Users\sura\OneDrive\Desktop\face_rec\venv\Scripts\python.exe C:/Users/sura/OneDrive/Desktop/face_rec/fsggsd.py

Suraj

Enter otp 4565555

Re-enter the otp 34455

Retry 1 attempts left

Re-enter the otp 297869

Access granted

Process finished with exit code 0

References :

1. <https://www.geeksforgeeks.org/python-face-recognition-using-gui/>
2. <https://pypi.org/project/face-recognition/>
3. <https://thecleverprogrammer.com/2021/04/14/otp-verification-using-python/>
4. <https://www.geeksforgeeks.org/python-program-to-generate-one-time-password-otp/>