**Common Part (for both Entry and Exit Code)**

**Importing Libraries**

import cv2

import numpy as np

import face\_recognition

import os

from datetime import datetime, timedelta

* **cv2 (OpenCV)**: A powerful library for image and video processing. Used to access the webcam, display images, and draw on the image.
* **numpy (np)**: A library for working with arrays. It is used here to handle numerical operations on images and face distances.
* **face\_recognition**: A Python library built on dlib that simplifies facial recognition tasks. It includes tools for encoding faces and comparing them.
* **os**: This standard Python library provides functions for interacting with the operating system, such as reading files and directories.
* **datetime & timedelta**: Libraries used to manipulate and calculate times and dates. Here, they help in timestamping entries/exits and comparing time differences.

**Loading Images of Known Students**

path = 'ImageAttendance'

images = []

classNames = []

myList = os.listdir(path)

* **path**: The folder where images of the students are stored.
* **images**: A list that will store the loaded images of students.
* **classNames**: A list that will store the names of the students, derived from the image filenames.
* **myList = os.listdir(path)**: Retrieves the names of all files in the ImageAttendance folder, which will be processed to load the images.

for cl in myList:

curImg = cv2.imread(f'{path}/{cl}')

images.append(curImg)

classNames.append(os.path.splitext(cl)[0])

* **cv2.imread(f'{path}/{cl}')**: Loads each image file in the folder. f'{path}/{cl}' is the file path of the current image.
* **images.append(curImg)**: Adds the loaded image (curImg) to the images list.
* **os.path.splitext(cl)[0]**: Extracts the filename without the extension (e.g., if image.jpg, it will return image). This is assumed to be the student's name, which is stored in classNames.

**Encoding the Faces**

def findEncodings(images):

encodeList = []

for img in images:

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

encodes = face\_recognition.face\_encodings(img)

if encodes:

encodeList.append(encodes[0])

return encodeList

* **Function Definition findEncodings()**: This function converts each image into a face encoding, which is a numerical representation of the face that can be compared with others to detect matches.
* **encodeList = []**: An empty list to store the encoded versions of the faces.
* **cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)**: Converts the color format of the image from BGR (OpenCV default) to RGB (the format expected by face\_recognition).
* **face\_recognition.face\_encodings(img)**: Finds the facial encodings in the image. If a face is detected, the encoding is returned as a list.
* **if encodes:**: To ensure that the face encoding exists (sometimes images may not have a detectable face).
* **encodeList.append(encodes[0])**: Adds the first detected face encoding to the list (assuming only one face per image).
* **return encodeList**: Returns the list of encoded faces.

encodeListKnown = findEncodings(images)

* Calls findEncodings() on the list of loaded images and stores the result (encoded faces) in encodeListKnown.

**Entry Code Specific Part**

**Tracking Last Entry Time for Each Student**

last\_entry\_time = {}

* **last\_entry\_time**: A dictionary that stores the last entry time for each student. The keys are the student's names, and the values are the last recorded timestamp.

**Marking Attendance and Recording Entry**

def markAttendance(name):

now = datetime.now()

* **markAttendance(name)**: This function records the student's entry in the Excel files. The student's name is passed as an argument.
* **now = datetime.now()**: Gets the current time when the entry is recorded.

**Time Gap Logic (15 Seconds Between Entries)**

if name in last\_entry\_time:

last\_time = last\_entry\_time[name]

if (now - last\_time).total\_seconds() < 15:

print(f"Skipping {name}'s entry. Less than 15 seconds since last entry.")

return

* **if name in last\_entry\_time:**: Checks if the student has an entry in the last\_entry\_time dictionary. This indicates whether the student has already entered recently.
* **last\_time = last\_entry\_time[name]**: Retrieves the last recorded entry time for the student.
* **if (now - last\_time).total\_seconds() < 15:**: Calculates the time difference between the current time (now) and the last entry time (last\_time). If it’s less than 15 seconds, the function exits without recording the entry.
* **print(f"Skipping {name}'s entry.")**: Prints a message indicating that the student's entry is skipped due to the time gap.

**Writing to Files**

last\_entry\_time[name] = now

* **last\_entry\_time[name] = now**: Updates the last entry time for the student to the current time.

with open('Attendance.csv', 'r+') as f:

myDataList = f.readlines()

nameList = [line.split(',')[0] for line in myDataList]

if name not in nameList:

f.writelines(f'\n{name},{dtString}')

* **open('Attendance.csv', 'r+')**: Opens the Attendance.csv file, which tracks students currently inside the hostel.
* **myDataList = f.readlines()**: Reads all the lines from the file.
* **nameList = [line.split(',')[0] for line in myDataList]**: Extracts the list of names currently inside the hostel.
* **if name not in nameList:**: If the student is not already in the hostel, their name is added.
* **f.writelines(f'\n{name},{dtString}')**: Writes the student’s name and the current time (dtString) to the file.

with open('Entry\_sheet.csv', 'a') as entry\_file:

entry\_file.write(f'{name},{dtString}\n')

* **open('Entry\_sheet.csv', 'a')**: Opens the Entry\_sheet.csv file in append mode. This file stores every entry event, irrespective of whether the student is already inside.
* **entry\_file.write(f'{name},{dtString}\n')**: Appends the student's name and the timestamp of entry to this file.

**Face Recognition Logic and Webcam Display**

cap = cv2.VideoCapture(0)

* **cv2.VideoCapture(0)**: Initializes the webcam capture, where 0 is the default webcam. This will start streaming video.

while True:

success, img = cap.read()

imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)

imgS = cv2.cvtColor(imgS, cv2.COLOR\_BGR2RGB)

* **cap.read()**: Reads the frame from the webcam. success tells if reading the frame was successful, and img is the captured frame.
* **cv2.resize(img, (0, 0), None, 0.25, 0.25)**: Resizes the image to a quarter of its original size for faster processing.
* **cv2.cvtColor(imgS, cv2.COLOR\_BGR2RGB)**: Converts the image color from BGR (default OpenCV format) to RGB, which is needed for face recognition.

facesCurFrame = face\_recognition.face\_locations(imgS)

encodesCurFrame = face\_recognition.face\_encodings(imgS, facesCurFrame)

* **face\_recognition.face\_locations(imgS)**: Detects the location of any faces in the current frame.
* **face\_recognition.face\_encodings(imgS, facesCurFrame)**: Generates face encodings for each detected face.

for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):

matches = face\_recognition.compare\_faces(encodeListKnown, encodeFace)

faceDis = face\_recognition.face\_distance(encodeListKnown, encodeFace)

matchIndex = np.argmin(faceDis)

* **zip(encodesCurFrame, facesCurFrame)**: Pairs the face encodings with the corresponding face locations.
* **face\_recognition.compare\_faces(encodeListKnown, encodeFace)**: Compares the detected face encoding with all known face encodings (students).
* **face\_recognition.face\_distance(encodeListKnown, encodeFace)**: Measures the distance between the detected