

SESSION 6

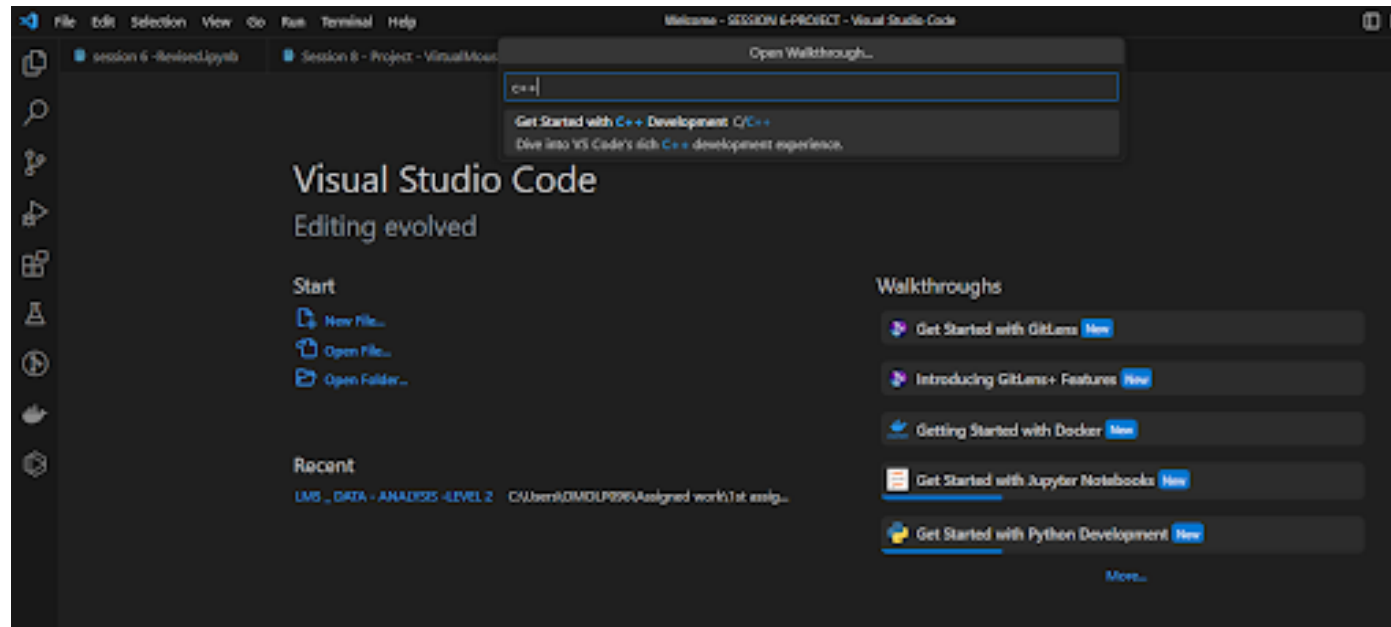
Project : Student attendance using face recognition

Importing Libraries

- `import face_recognition`: This imports the `face_recognition` library, which provides face detection and recognition capabilities.
- `import cv2`: This imports the OpenCV library, which is used for image and video processing tasks.
- `import numpy as np`: This imports the NumPy library, which provides support for multi-dimensional arrays and mathematical operations on them.
- `import csv`: This imports the CSV library, which allows reading and writing CSV files.
- `import os`: This imports the `os` module, which provides functions for interacting with the operating system, such as file and directory operations.
- `from datetime import datetime`: This imports the `datetime` class from the `datetime` module, which allows working with dates and times.

Steps - Installing face_recognition Library

Step 1 : Install the C++ Module in VS Code IDE as shown below



Step 2 : Follow the steps to complete the installation

Install a C++ compiler on Windows

If you're doing C++ development for Windows, we recommend installing the Microsoft Visual C++ (MSVC) compiler toolset. If you're targeting Linux from Windows, check out [Using C++ and Windows Subsystem for Linux \(WSL\) in VS Code](#). Or, you could [install GCC on Windows with MinGW](#).

1. To install MSVC, download **Build Tools for Visual Studio 2022** from the Visual Studio [Downloads](#) page.
2. In the Visual Studio Installer, check the **C++ build tools** workload and select **Install**.

Note: You can use the C++ toolset from Visual Studio Build Tools along with Visual Studio Code to compile, build, and verify any C++ codebase as long as you also have a valid Visual Studio license (either Community, Pro, or Enterprise) that you are actively using to develop that C++ codebase.

3. Open the **Developer Command Prompt for VS** by typing 'developer' in the Windows Start menu.
4. Check your MSVC installation by typing `cl` into the Developer Command Prompt for VS. You should see a copyright message with the version and basic usage description.

Note: To use MSVC from the command line or VS Code, you must run from a **Developer Command Prompt for VS**. An ordinary shell such as PowerShell, Bash, or the Windows command prompt does not have the necessary path environment variables set.

Other option is to Download using the link below:

<https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2022>

Build Tools for Visual
Studio 2022

These Build Tools allow you to build Visual Studio projects from a command-line interface. Supported projects include: ASP.NET, Azure, C++ desktop, ClickOnce, containers, .NET Core, .NET Desktop, Node.js, Office and SharePoint, Python, TypeScript, Unit Tests, UWP, WCF, and Xamarin. Use of this tool requires a valid Visual Studio license, unless you are building open-source dependencies for your project. See the [Build Tools license](#) for more details.

Are you looking for one of the Visual Studio 2022 [long term servicing baselines \(LTSCs\)](#)? You can find them [here](#).

Download

Visual Studio Installer

Installed Available

All installations are up to date.



Visual Studio Community 2022

17.6.2

Powerful IDE, free for students, open-source contributors, and individuals

[Release notes](#)

Modify

Launch

More ▼

Modifying — Visual Studio Professional 2019 — 16.3.4

Workloads Individual components Language packs Installation locations

Web & Cloud (4)

- ☐ **ASP.NET and web development**
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker support.
- ☐ **Python development**
Editing, debugging, interactive development and source control for Python.
- ☐ **Azure development**
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET Core and .NET...
- ☐ **Node.js development**
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Windows (3)

- ☐ **.NET desktop development**
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET Core and .NET...
- ☒ **Desktop development with C++**
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- ☐ **Universal Windows Platform development**
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Installation details

- > Visual Studio core editor
- ✓ Desktop development with C++ *
 - Included
 - ✓ C++ core desktop features
 - Optional
 - ☒ MSVC v142 - VS 2019 C++ x64/x86 build tools (...)
 - ☒ Windows 10 SDK (10.0.18362.0)
 - ☒ Just-In-Time debugger
 - ☒ C++ profiling tools
 - ☒ C++ CMake tools for Windows
 - ☒ C++ ATL for latest v142 build tools (x86 & x64)
 - ☒ Test Adapter for Boost.Test
 - ☒ Test Adapter for Google Test
 - ☒ Live Share
 - ☒ IntelliCode
 - ☐ C++ MFC for latest v142 build tools (x86 & x64)
 - ☐ C++/CLI support for v142 build tools (14.23)
 - ☐ C++ Modules for v142 build tools (x64/x86 - ex...
 - ☐ C++ Clang tools for Windows (8.0.1 - x64/x86)
 - ☐ IncrediBuild - Build Acceleration

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional

Total space required 1.24 GB

By continuing, you agree to the [license](#) for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the [3rd Party Notices](#) or in its accompanying license. By continuing, you also agree to those licenses.

Install while downloading Modify

Python Installation Modules

Please follow the below sequence of Install:

- pip install cmake
- pip install wheel
- pip install dlib --user
- pip install protobuf
- pip install face_recognition

```
In [ ]: !pip install cmake
!pip install wheel
!pip install dlib --user
!pip install protobuf
!pip install face_recognition
```

Collecting cmake

Using cached cmake-3.27.7-py2.py3-none-win_amd64.whl (34.6 MB)

Installing collected packages: cmake

Successfully installed cmake-3.27.7

Requirement already satisfied: wheel in c:\users\omolp091\anaconda3\lib\site-packages (0.38.4)

Requirement already satisfied: dlib in c:\users\omolp091\anaconda3\lib\site-packages (19.24.2)

Requirement already satisfied: protobuf in c:\users\omolp091\anaconda3\lib\site-packages (3.20.3)

Requirement already satisfied: face_recognition in c:\users\omolp091\anaconda3\lib\site-packages (1.3.0)

Requirement already satisfied: dlib>=19.7 in c:\users\omolp091\anaconda3\lib\site-packages (from face_recognition) (19.24.2)

Requirement already satisfied: Pillow in c:\users\omolp091\anaconda3\lib\site-packages (from face_recognition) (9.4.0)

Requirement already satisfied: face-recognition-models>=0.3.0 in c:\users\omolp091\anaconda3\lib\site-packages (from face_recognition) (0.3.0)

Requirement already satisfied: Click>=6.0 in c:\users\omolp091\anaconda3\lib\site-packages (from face_recognition) (8.0.4)

Requirement already satisfied: numpy in c:\users\omolp091\anaconda3\lib\site-packages (from face_recognition) (1.23.5)

Requirement already satisfied: colorama in c:\users\omolp091\anaconda3\lib\site-packages (from Click>=6.0->face_recognition) (0.4.6)

Python Imports

```
In [ ]: import face_recognition
import cv2
import numpy as np
import csv
import os
from datetime import datetime
from PIL import Image as im
```

- The line `video_capture = cv2.VideoCapture(0)` initializes the `video_capture` object, which represents the video capturing device. In this case, 0 as the argument specifies that the default camera should be used for capturing video.

```
In [ ]: video_capture = cv2.VideoCapture(0)
```

- In summary, the code loads images of known faces and extracts their face features using the `face_recognition` library. These encoded features can later be used for face recognition tasks, such as comparing them with faces detected in a live video stream.

```
In [ ]: # Load the images of known faces and encode their face features
jobs_image = face_recognition.load_image_file("photos/jobs.jpg")
jobs_encoding = face_recognition.face_encodings(jobs_image)[0]
face_location = face_recognition.face_locations(jobs_image)

ratan_tata_image = face_recognition.load_image_file("photos/tata.jpg")
ratan_tata_encoding = face_recognition.face_encodings(ratan_tata_image)[0]

kohli_image = face_recognition.load_image_file("photos/kohli.jpg")
kohli_encoding = face_recognition.face_encodings(kohli_image)[0]

tesla_image = face_recognition.load_image_file("photos/tesla.jpg")
tesla_encoding = face_recognition.face_encodings(tesla_image)[0]
```

a. `known_face_encoding`: This list contains the face encodings of known faces. The encodings are stored in the order of the corresponding names in the `known_faces_names` list. In this case, the `jobs_encoding`, `ratan_tata_encoding`, `kohli_encoding`, and `tesla_encoding` variables (which were defined in the previous code snippet) are used to populate this list. These encodings represent the facial features of Steve Jobs, Ratan Tata, Virat Kohli, and an image related to Tesla, respectively.

b. `known_faces_names`: This list contains the names corresponding to the known face encodings. The names are provided as strings and are arranged in the same order as the face encodings in the `known_face_encoding` list. In this case, the names are "jobs", "ratan tata", "kohli", and "tesla". These names identify the individuals associated with the respective face encodings.

By organizing the face encodings and names in these lists, you can easily associate a detected face's encoding with its corresponding name during the face recognition process.

```
In [ ]: # Create lists of known face encodings and corresponding names
known_face_encoding = [
```

```
jobs_encoding,  
ratan_tata_encoding,  
kohli_encoding,  
tesla_encoding  
]  
  
known_faces_names = [  
    "jobs",  
    "ratan tata",  
    "kohli",  
    "tesla"  
]
```

The variable `students` is a copy of a list called `known_faces_names`. It is assumed that `known_faces_names` contains the names of students whose faces are known.

The variables `face_locations`, `face_encodings`, and `face_names` are empty lists, which will be populated later in the code.

The variable `s` is set to `True`, but its purpose is not clear from the given code snippet.

The `datetime.now()` function is used to get the current date and time, and the `strftime()` method formats it to only include the year, month, and day in the format "YYYY-MM-DD". This formatted date is stored in the variable `current_date`.

A CSV file is then opened with the filename being the current date concatenated with ".csv". The file is opened in write mode, and if the file does not exist, it will be created.

A `csv.writer` object named `Inwriter` is created, which will be used to write data to the CSV file.

```
In [ ]: students = known_faces_names.copy()  
  
face_locations = []  
face_encodings = []  
face_names = []  
s=True  
  
now = datetime.now()  
current_date = now.strftime("%Y-%m-%d")
```

```
f = open(current_date+'.csv','w+',newline = '')  
lnwriter = csv.writer(f)
```

The next step involves detecting faces in the frame and encoding their features. The face_recognition library is used for this purpose. The face_locations variable stores the locations of the detected faces in the smaller frame, while the face_encodings variable contains the corresponding face encodings.

Finally, the face_names list is initialized to store the names associated with the detected faces. This list will be populated later in the code, possibly by matching the face encodings with a known set of encodings. The loop continues indefinitely, processing each frame in a similar manner.

```
In [ ]: while True:  
    _, frame = video_capture.read()  
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)  
    rgb_small_frame = frame[:, :, :-1]  
  
    # Detect faces in the frame and encode their features  
    face_location = face_recognition.face_locations(rgb_small_frame)  
    face_encoding = face_recognition.face_encodings(rgb_small_frame.astype('uint8'), face_location)  
    face_names = []
```

For each face encoding in the face_encodings list, the code compares it with the known_face_encoding list using the compare_faces function from the face_recognition library. The result is stored in the matches list, where each element corresponds to a match or non-match for a known face.

The code then initializes an empty string variable name to store the recognized person's name. It also calculates the face distance between the detected face and each known face using the face_distance function. The index of the best match is obtained by finding the index of the smallest face distance using np.argmin.

If there is a match (matches[best_match_index] is True), the recognized person's name is assigned from the known_faces_names list based on the index of the best match. Finally, the recognized name is appended to the face_names list.

This process allows the code to compare the face encodings of the detected faces with a known set of face encodings and assign names to the recognized faces, populating the face_names list accordingly.

```
In [ ]: for face_encoding in face_encodings:  
    matches = face_recognition.compare_faces(known_face_encoding, face_encoding)  
    name = ""  
    face_distance = face_recognition.face_distance(known_face_encoding, face_encoding)
```



```
best_match_index = np.argmin(face_distance)

if matches[best_match_index]:
    name = known_faces_names[best_match_index]

face_names.append(name)
```

After appending the recognized name to the face_names list, the code checks if the name exists in the known_faces_names list. This check ensures that only known faces are considered for further processing.

If the name is found in the known_faces_names list, the code proceeds to draw text on the frame indicating the presence of the recognized person. It uses the putText function from OpenCV to add text to the frame.

The parameters for the putText function are as follows:

frame: The frame on which the text will be drawn. name+' Present': The text to be displayed, indicating the presence of the recognized person. bottomLeftCornerOfText: The coordinates of the bottom-left corner of the text box. font: The font type. fontScale: The scale of the font. fontColor: The color of the font, specified as a tuple of RGB values. thickness: The thickness of the font stroke. lineType: The type of line used to draw the font. The code then checks if the recognized name is present in the students list, and if so, it removes the name from the list and prints the updated students list. This step suggests that the recognized person is a student and their attendance is being recorded.

Next, the code gets the current time using the now.strftime("%H-%M-%S") function, which formats the time as "hours-minutes-seconds". It writes the name and current time to the CSV file using the Inwriter.writerow([name,current_time]) function, presumably for attendance tracking purposes.

Finally, the code displays the frame with the added text using cv2.imshow("attendance system",frame). It also checks for the "q" key press and if detected, breaks the loop to exit the program.

```
In [ ]: face_names.append(name)
if name in known_faces_names:
    font = cv2.FONT_HERSHEY_SIMPLEX
    bottomLeftCornerOfText = (10,100)
    fontScale = 1.5
    fontColor = (255,0,0)
    thickness = 3
    lineType = 2

    cv2.putText(frame,name+' Present',
                bottomLeftCornerOfText,
                font,
```

```

        fontScale,
        fontColor,
        thickness,
        lineType)

    if name in students:
        students.remove(name)
        print(students)
        current_time = now.strftime("%H-%M-%S")
        lnwriter.writerow([name, current_time])
    cv2.imshow("attendance system", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

```

video_capture.release(): This function releases the resources associated with the video capture. It is necessary to release the video capture when it is no longer needed to free up system resources.

cv2.destroyAllWindows(): This function closes all windows created by OpenCV. It is useful to ensure that all windows are closed properly before the program exits.

f.close(): This function closes the file object f that was used to write data to a file, presumably the CSV file for attendance records. It is important to close the file after writing to ensure that all data is properly saved and that system resources are released.

```

In [ ]: video_capture.release()
        cv2.destroyAllWindows()
        f.close()

```

Complete code

```

In [ ]: import face_recognition
import cv2
import numpy as np
import csv
import os
from datetime import datetime
from PIL import Image as im

video_capture = cv2.VideoCapture(0)

jobs_image = face_recognition.load_image_file("photos/jobs.jpg")

```

```
jobs_encoding = face_recognition.face_encodings(jobs_image)[0]

ratan_tata_image = face_recognition.load_image_file("photos/tata.jpg")
ratan_tata_encoding = face_recognition.face_encodings(ratan_tata_image)[0]

kohli_image = face_recognition.load_image_file("photos/kohli.jpg")
kohli_encoding = face_recognition.face_encodings(kohli_image)[0]

radhakrishnan_image = face_recognition.load_image_file("photos/radhakrishnan.jpg")
radhakrishnan_encoding = face_recognition.face_encodings(radhakrishnan_image)[0]

known_face_encoding = [
    jobs_encoding,
    ratan_tata_encoding,
    kohli_encoding,
    radhakrishnan_encoding
]

known_faces_names = [
    "jobs",
    "ratan tata",
    "kohli",
    "radhakrishnan"
]

students = known_faces_names.copy()

face_locations = []
face_encodings = []
face_names = []
s=True

now = datetime.now()
current_date = now.strftime("%Y-%m-%d")

f = open(current_date+'.csv','w+',newline = '')
lnwriter = csv.writer(f)

while True:
    _,frame = video_capture.read()
    # im.fromarray(frame).show()
    small_frame = cv2.resize(frame,(0,0),fx=0.25,fy=0.25)
```

```

# im.fromarray(small_frame).show()
rgb_small_frame = small_frame[:, :, :-1]
# im.fromarray(rgb_small_frame).show()
if s:
    face_location1 = face_recognition.face_locations(rgb_small_frame)
    face_encoding1 = face_recognition.face_encodings(rgb_small_frame.astype('uint8'), face_location1)
    face_names = []
    for face_encoding, face_location in zip(face_encoding1, face_location1):
        matches = face_recognition.compare_faces(known_face_encoding, face_encoding)
        # print(matches)
        name = ""
        face_distance = face_recognition.face_distance(known_face_encoding, face_encoding)
        best_match_index = np.argmin(face_distance)
        if matches[best_match_index]:
            name = known_faces_names[best_match_index]

    face_names.append(name)
    if name in known_faces_names:
        font = cv2.FONT_HERSHEY_SIMPLEX
        bottomLeftCornerOfText = (10, 100)
        fontScale = 1.5
        fontColor = (255, 0, 0)
        thickness = 3
        lineType = 2

        cv2.putText(frame, name + ' Present',
                    bottomLeftCornerOfText,
                    font,
                    fontScale,
                    fontColor,
                    thickness,
                    lineType)

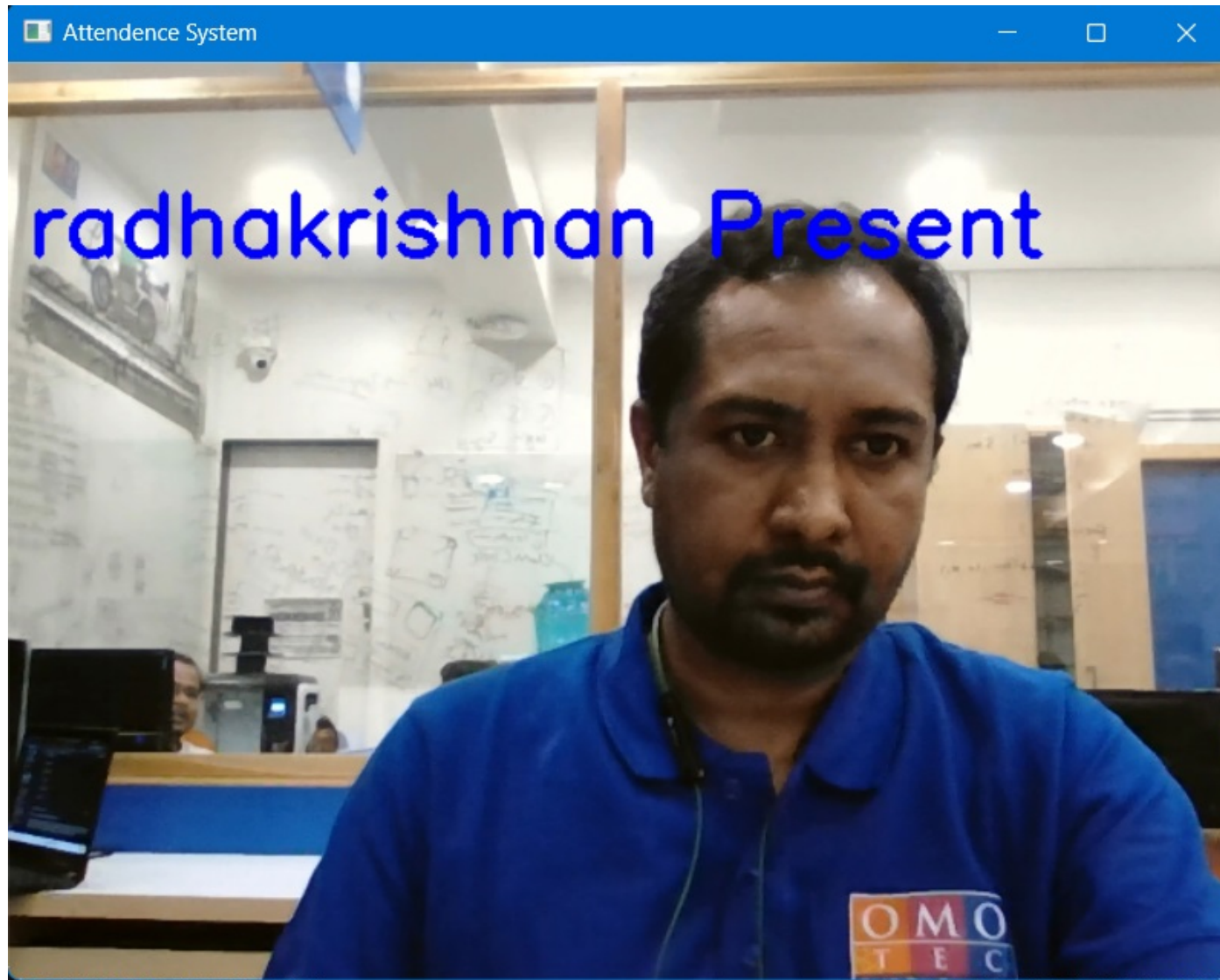
        if name in students:
            students.remove(name)
            print(students)
            current_time = now.strftime("%H-%M-%S")
            lnwriter.writerow([name, current_time])
    cv2.imshow("Attendance System", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


video_capture.release()

```

```
cv2.destroyAllWindows()  
f.close()
```

```
['jobs', 'ratan tata', 'kohli']
```



SESSION 6-PROJECT new >  2023-12-02.csv

1 |radhakrishnan,18-21-02

2