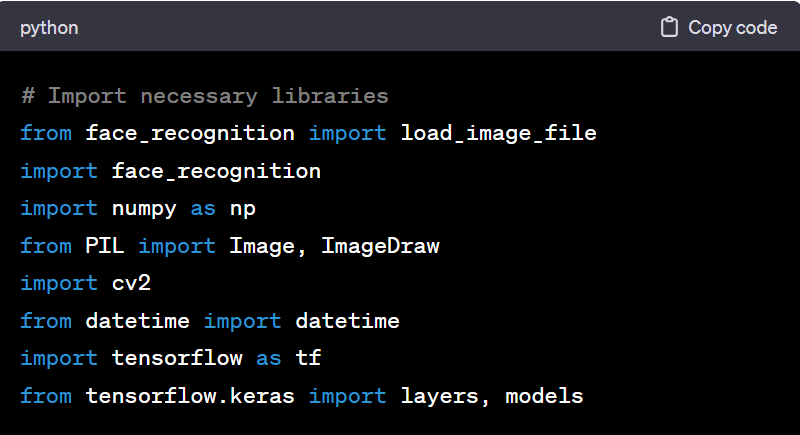
Code Explanation



1.**face\_recognition**: Used for face recognition and encoding.

->Face Detection: The library can locate faces in images and provide the coordinates of the face bounding boxes.

->Face Encoding: It can generate numerical encodings for faces, which can be used for face comparison and recognition.

-> Face Comparison: Given face encodings, the library can compare faces to determine if they are the same or different.

->Face Recognition: It allows you to recognize known faces in images or video streams by comparing them with a database of known face encodings.

2. **numpy :** Used for numerical operations.

->To perform a wide variety of mathematical operations on arrays.

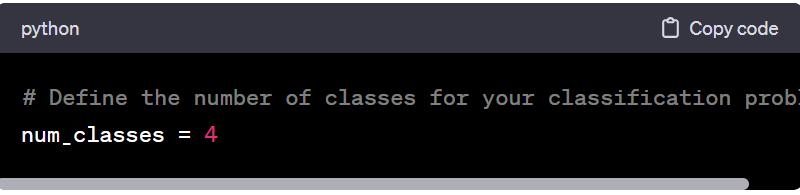
3. **PIL (Pillow):** Used for working with images. contains all the basic image processing functionality. You can do image resizing, rotation, and transformation.

4. **cv2 (OpenCV):** Used for computer vision tasks, such as capturing video frames and drawing on images.To display an image in a window.

-> OpenCV (cv2) is a powerful open-source library for computer vision and image processing, providing tools for tasks like image manipulation, feature detection, and machine learning integration. It is widely used in fields such as robotics, automation, and computer vision research.

5. **datetime:** Used for working with dates and times.

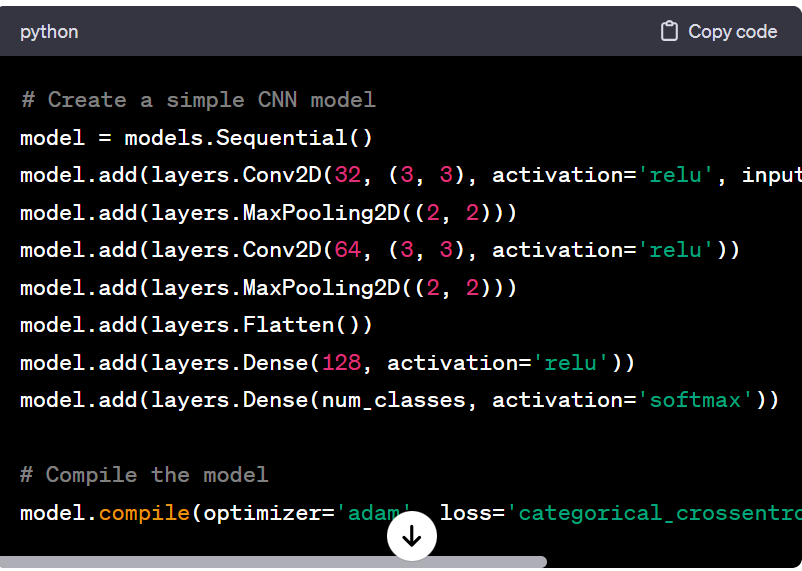
6. **TensorFlow:** Used for creating and training neural networks. TensorFlow is used to create and train a simple Convolutional Neural Network (CNN) for a face recognition task.

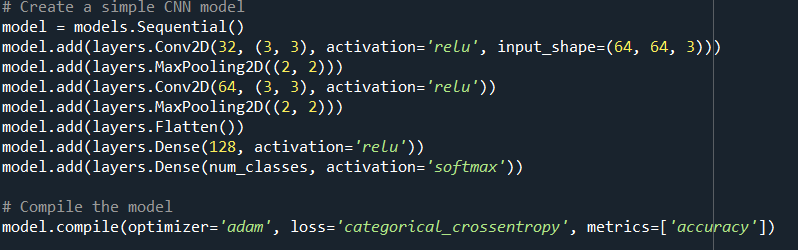


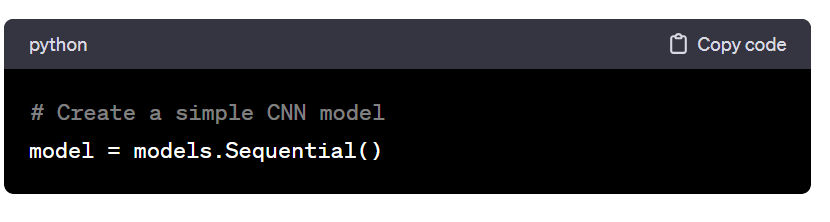
**‘num\_classes’** represents the number of classes in your classification problem. In this case, it is set to 4, indicating that there are four classes (faces) to be recognized.

Or

-> ‘**num\_classes’** is set to **4**, indicating that the classification problem involves distinguishing between four different categories or classes. It is crucial for designing and training machine learning models.







**Model Initialization:**

**model**: This variable represents the neural network model. It is initialized as a sequential model, which means that layers are added one after the other.



**[Convolution: Convolution puts the image through a set of filters(kernels). Each filter activates certain features in the image, relevant to the learning task.](https://www.tensorflow.org/)**

**[• Ex: if the model has to detect faces in the image, there may be filters for edge detection, shape detection, etc moving through the image to capture those significant images for face detection](https://www.tensorflow.org/)**

**Convolutional Layer (1):**

* **layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(64, 64, 3))**: This adds the first convolutional layer to the model.
* **32**: The number of filters (also known as kernels or channels) in the convolutional layer.
* **(3, 3)**: The size of each filter in the convolutional layer.
* **activation='relu'**: The Rectified Linear Unit (ReLU) activation function, which introduces non-linearity to the model.
* **input\_shape=(64, 64, 3)**: The shape of the input data. In this case, it's a 3D tensor with dimensions 64x64 pixels and 3 channels (representing RGB color).



**Max Pooling Layer**

**layers.MaxPooling2D((2, 2))**: This adds the first max-pooling layer to the model. Max pooling reduces the spatial dimensions of the data, helping the model focus on the most important features.



* **Rectified linear unit (ReLU)** allows for faster and more effective training by mapping negative values to zero and maintaining positive values.

**Convolutional Layer**

**layers.Conv2D(64, (3, 3), activation='relu')**: This adds the second convolutional layer to the model with 64 filters



**Max Pooling Layer (2):**

* **layers.MaxPooling2D((2, 2))**: This adds the second max-pooling layer to the model.



**model.add(layers.Flatten())** adds a flatten layer to the neural network model, converting the multi-dimensional output from previous layers into a one-dimensional vector, preparing it for connection to fully connected (dense) layers.



**Dense Layer (1):**

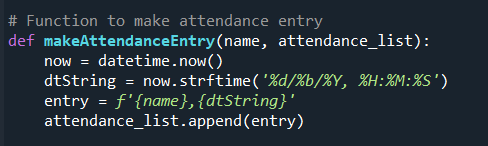
* **layers.Dense(128, activation='relu')**: This adds a dense layer with 128 neurons and ReLU activation. Dense layers are fully connected layers.



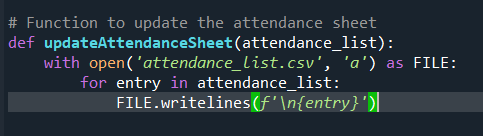
The output layer with a number of neurons equal to the number of classes (‘**num\_classes’**). The softmax activation function is used for multi-class classification problems, providing probability scores for each class.



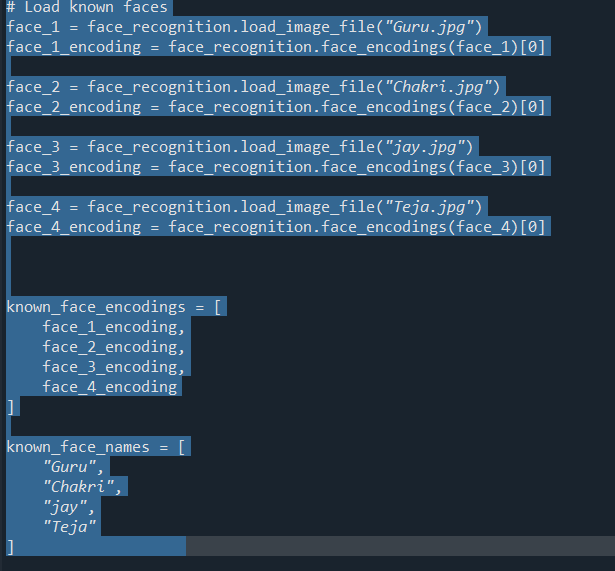
* **model.compile**: This function configures the model for training.
* **optimizer='adam'**: The Adam optimizer, an optimization algorithm.
* **loss='categorical\_crossentropy'**: The categorical cross-entropy loss function, commonly used for multi-class classification problems.
* **metrics=['accuracy']**: The metric used to evaluate the model's performance during training and testing is accuracy.
* This code defines a simple CNN architecture for a classification problem with two convolutional layers, max-pooling layers, a flatten layer, and two dense layers.



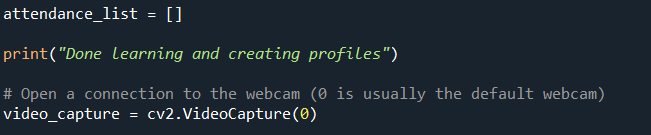
**‘makeAttendanceEntry’** takes a name and an attendance list as input, gets the current date and time, and creates an entry with the format 'name, date/time'. This entry is then appended to the attendance list.



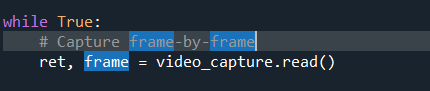
This function **updateAttendanceSheet** takes an attendance list as input and appends the entries to a CSV file named 'attendance\_list.csv'.



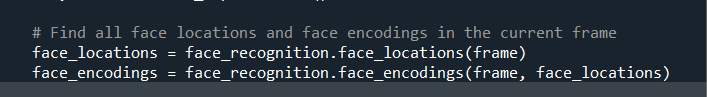
Known faces and their corresponding encodings are loaded and stored in lists **known\_face\_encodings** and **known\_face\_names**.



An empty attendance list is created, and the code establishes a connection to the default webcam (index 0).

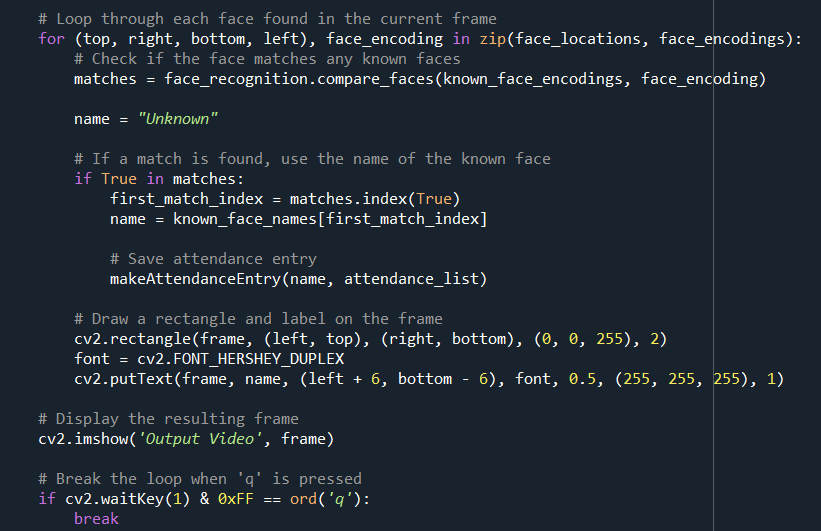


This starts an infinite loop where each iteration captures a frame from the webcam.

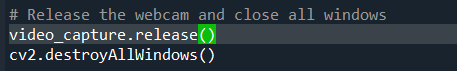


1. **face\_locations = face\_recognition.face\_locations(frame)**:
   * **face\_recognition.face\_locations()** is a function that returns an array of bounding box coordinates for the faces present in the given image or frame.
   * Here, it is applied to the **frame** obtained from the webcam feed, and the resulting **face\_locations** variable stores a list of tuples. Each tuple contains the coordinates **(top, right, bottom, left)** of the bounding box for a detected face.
2. **face\_encodings = face\_recognition.face\_encodings(frame, face\_locations)**:
   * **face\_recognition.face\_encodings()** is a function that computes the face encodings (128-dimensional vectors) for the faces detected in the given image or frame.
   * It takes the **frame** and the previously obtained **face\_locations** as input and returns a list of 128-dimensional face encodings.
   * The resulting **face\_encodings** variable stores these encodings for each detected face.

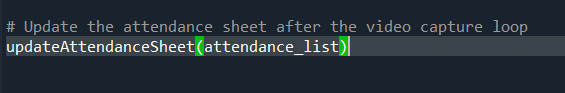
These two steps are crucial for face recognition. The face locations provide the position of faces in the frame, and the face encodings provide a compact representation of facial features, which can be compared with the known faces' encodings to determine if there is a match. Later in the code, these face locations and encodings are used to compare with the known faces' data and update the attendance list if a match is found.



This loop iterates through each face in the current frame, compares it with known faces, updates the attendance list, and draws rectangles and labels on the frame accordingly.

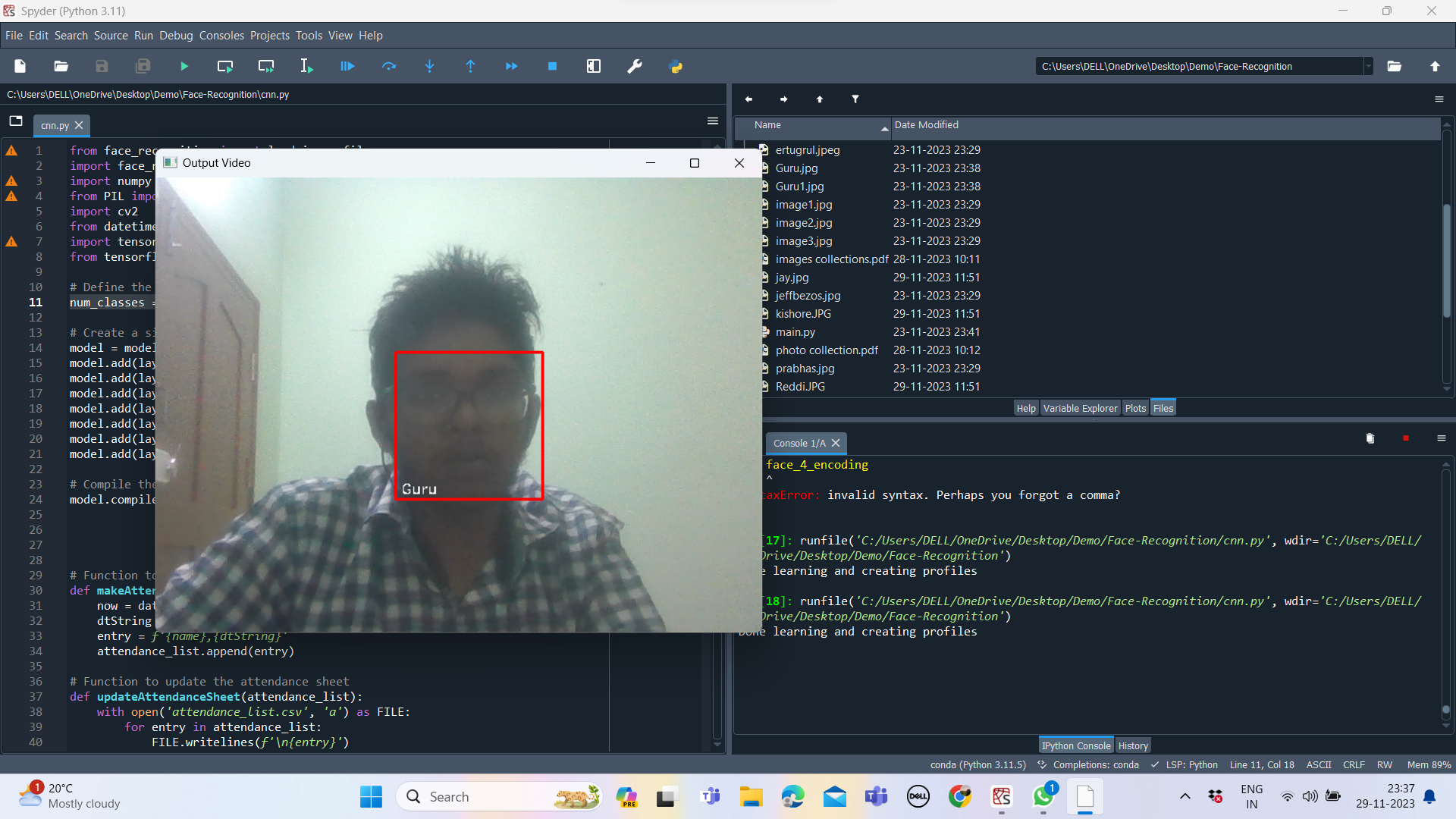


After breaking out of the loop, the code releases the webcam and closes all OpenCV windows.



Finally, the attendance list is updated to the CSV file after the video capture loop.

**OUTPUT**

****