Test.py file:

```
from sklearn.neighbors import KNeighborsClassifier
import cv2
import pickle
import numpy as np
import os
import csv
import time
from datetime import datetime
from win32com.client import Dispatch
```

These are import statements to import necessary libraries/modules like scikit-learn's KNeighborsClassifier for face recognition, OpenCV for video capturing, pickle for object serialization, numpy for mathematical computations, os for file management, csv for file I/O, time for timestamping, datetime for date formatting, and Dispatch for text-to-speech conversion.

Import pickle: The `pickle` module in Python is used to serialize and deserialize Python objects. This means that `pickle` allows you to convert a Python object into a byte stream that can be stored in a file, transmitted over a network, or sent between processes, and then reconstruct the original object from that byte stream later. The process of converting a Python object into a byte stream is called pickling, and the process of reconstructing the original object from the byte stream is called unpickling. `pickle` is commonly used to save and load machine learning models, as well as to save and load other complex data structures in Python.

Import cv2: OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It was originally developed by Intel and is now maintained by the OpenCV community. OpenCV provides a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. It is used for a wide range of tasks such as image and video processing, object detection and recognition, face recognition, and more. The library is written in C++ and provides interfaces for multiple programming languages including Python, Java, and MATLAB.

```
def speak(str1):
    speak=Dispatch(("SAPI.SpVoice"))
    speak.Speak(str1)
```

This function is used for text-to-speech conversion. It takes a string as input and outputs the audio of the input text using the Windows Speech API.

```
video=cv2.VideoCapture(0)
facedetect=cv2.CascadeClassifier('data/haarcascade_frontalface_default.xml')
```

This code initializes the video capture object and creates a face detection object using OpenCV's CascadeClassifier class.

```
with open('data/names.pkl', 'rb') as w:
    LABELS=pickle.load(w)
with open('data/faces_data.pkl', 'rb') as f:
    FACES=pickle.load(f)
```

These lines load the saved face recognition model and label data from binary files using pickle module.

```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(FACES, LABELS)
```

These lines initialize the KNN classifier and train it using the face data and corresponding labels.

fit(): In scikit-learn, for example, `fit()` is a method used on a model object to train the model on the given data. It takes as input the training data (usually a feature matrix X and target variable y) and learns a set of parameters that allow the model to make predictions on new data.

For example, if we have a classification problem where we want to predict whether a given email is spam or not, we might use a logistic regression model. We would first split our data into a training set and a test set, and then call the `fit()` method on the training data to train the model. The model would learn a set of weights that allow it to make predictions on new data, and we could then evaluate the model's performance on the test set using the `predict()` method.

```
imgBackground=cv2.imread("background.png")
```

This code reads an image file "background.png" which is used as a background for displaying the captured video frames.

COL_NAMES = ['NAME', 'TIME']

This line creates a list of column names to be used in the attendance CSV file.

while True:
ret,frame=video.read()
This starts an infinite loop where the frames are read from the video capture device (e.g., a webcam) using the `video.read()` method. The returned value `ret` indicates whether a frame was successfully read or not, while `frame` contains the captured frame.

gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
This converts the captured frame to grayscale using the `cv2.cvtColor()` method. This is done because the k-NN algorithm works better with grayscale images.

grayscale: This converts the captured frame to grayscale using the `cv2.cvtColor()` method. This is done because the k-NN algorithm works better with grayscale images.

faces=facedetect.detectMultiScale(gray, 1.3 ,5)

This detects faces in the grayscale image using the Haar cascade classifier, `facedetect`. The `detectMultiScale()` method takes the grayscale image as input, as well as two parameters: `scaleFactor` and `minNeighbors`. The `scaleFactor` is used to reduce the image size at each image scale, while `minNeighbors` specifies how many neighbors each candidate rectangle should have in order to be retained as a face. The output is a list of tuples representing the detected faces, where each tuple contains the `(x,y,w,h)` coordinates of the face rectangle.

for (x,y,w,h) in faces:
This starts a loop over the detected faces, where `(x,y,w,h)` are the coordinates of the face rectangle.
crop_img=frame[y:y+h, x:x+w, :]
This extracts the face region from the original color image using NumPy array slicing.

resized_img=cv2.resize(crop_img, (50,50)).flatten().reshape(1,-1)

This resizes the face region to a fixed size of 50x50 pixels and flattens it into a 1D NumPy array using the `flatten()` method. It then reshapes it into a 2D array of shape `(1, 2500)` using the `reshape()` method. This is done to make it compatible with the training data for the k-NN algorithm.

```
        output=knn.predict(resized_img)
```

This uses the trained k-NN classifier, `knn`, to predict the label of the input face image, `resized_img`. The predicted label, `output`, is a string representing the name of the recognized person.

```
        ts=time.time()
        date=datetime.fromtimestamp(ts).strftime("%d-%m-%Y")
        timestamp=datetime.fromtimestamp(ts).strftime("%H:%M-%S")
```

This gets the current timestamp and formats it into a date and time string using the `datetime.fromtimestamp()` method.

```
        exist=os.path.isfile("Attendance/Attendance_" + date + ".csv")
```

This checks if a CSV file for the current date already exists in the "Attendance" folder using the `os.path.isfile()` method.

```
        cv2.rectangle(frame, (x,y), (x+w, y+h), (0,0,255), 1)
        cv2.rectangle(frame,(x,y),(x+w,y+h),(50,50,255),2)
        cv2.rectangle(frame,(x,y-40),(x+w,y),(50,50,255),-1)
        cv2.putText(frame, str(output[0]), (x,y-15), cv2.FONT_HERSHEY_COMPLEX, 1,
(255,255,255), 1)
        cv2.rectangle(frame, (x,y), (x+w, y+h), (50,50,255), 1)
```

This draws a rectangle around the detected face in the original color image using the `cv2.rectangle()` method.

These lines draw rectangles around the detected face in the original color frame to highlight it. The `cv2.rectangle()` function is used to draw a rectangle around the face region of interest (ROI) and then another rectangle is drawn around it with a thicker border. Another rectangle is drawn above the face rectangle with a background color to hold the text. The predicted output is then displayed on the frame using the `cv2.putText()` function.

```
        attendance=[str(output[0]), str(timestamp)]
    imgBackground[162:162 + 480, 55:55 + 640] = frame
    cv2.imshow("Frame", imgBackground)
    k=cv2.waitKey(1)
```

The predicted output and the timestamp are stored in the `attendance` list. The frame with the highlighted face and the predicted output is then overlaid onto a background image using array slicing. The background image is then displayed using the `cv2.imshow()` function.

```
        if k==ord('o'):
        speak("Attendance Taken..")
        time.sleep(5)
        if exist:
            with open("Attendance/Attendance_" + date + ".csv", "+a") as csvfile:
                writer=csv.writer(csvfile)
                writer.writerow(attendance)
            csvfile.close()
        else:
            with open("Attendance/Attendance_" + date + ".csv", "+a") as csvfile:
                writer=csv.writer(csvfile)
                writer.writerow(COL_NAMES)
                writer.writerow(attendance)
            csvfile.close()
    if k==ord('q'):
        break
video.release()
cv2.destroyAllWindows()
```

This section of the code is responsible for taking attendance and saving it in a CSV file. The function `cv2.waitKey(1)` waits for a key event to happen for 1 millisecond. If any key is pressed, it is stored in variable `k`. If the key pressed is 'o' then the attendance is taken. The system speaks out "Attendance Taken.." and then waits for 5 seconds. After this, it checks if the attendance file exists or not. If it exists, it opens the file in append mode and writes the `attendance` data. Otherwise, it creates a new file, writes the column names (`COL_NAMES`) and then writes the `attendance` data. If the key pressed is 'q' then the loop is broken and the program is terminated. Finally, the video is released and all windows are closed.

Add_faces File

```
import cv2
import pickle
import numpy as np
import os
video=cv2.VideoCapture(0)
```

This line creates a VideoCapture object that captures video from the default camera (0).

```
facedetect=cv2.CascadeClassifier('data/haarcascade_frontalface_default.xml')
```

This line creates a CascadeClassifier object from the XML file that contains the parameters for detecting faces.

```
faces_data=[]
```

his initializes an empty list that will be used to store the face data.

```
i=0
```

This initializes a counter variable that will be used to track the number of faces detected

```
name=input("Enter Your Name: ")
```

this line prompts the user to enter their name.

```
while True:
    ret,frame=video.read()
    gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces=facedetect.detectMultiScale(gray, 1.3 ,5)
    for (x,y,w,h) in faces:
        crop_img=frame[y:y+h, x:x+w, :]
        resized_img=cv2.resize(crop_img, (50,50))
        if len(faces_data)<=100 and i%10==0:
            faces_data.append(resized_img)
        i=i+1
        cv2.putText(frame, str(len(faces_data)), (50,50),
cv2.FONT_HERSHEY_COMPLEX, 1, (50,50,255), 1)
        cv2.rectangle(frame, (x,y), (x+w, y+h), (50,50,255), 1)
    cv2.imshow("Frame",frame)
    k=cv2.waitKey(1)
    if k==ord('q') or len(faces_data)==100:
        break
```

These lines set up a loop that reads the video frame-by-frame and detects faces. For each detected face, it crops the image and resizes it to 50x50. If the number of faces detected is less than or equal to 100 and the current index is a multiple of 10, the face data is appended to the `faces_data` list. It then draws a rectangle around the detected face and displays the number of faces detected on the screen. The loop continues until either 100 faces have been detected or the user presses the 'q' key.

```
video.release()
cv2.destroyAllWindows()
```

These lines release the video capture object and close all windows.

```
faces_data=np.asarray(faces_data)
faces_data=faces_data.reshape(100, -1)
```

These lines convert the `faces_data` list to a NumPy array and reshape it to have 100 rows and a single column.

These lines create or update a binary file called 'names.pkl' that contains the names of the people whose faces were detected. If the file doesn't exist, it creates a list with the user's name repeated 100 times and saves it to the file. If the file already exists, it loads the existing list, appends the user's name to it 100 times,

```
if 'names.pkl' not in os.listdir('data/'):
    names=[name]*100
    with open('data/names.pkl', 'wb') as f:
        pickle.dump(names, f)
```

- The `if` statement checks if a file named 'names.pkl' exists in the 'data/' directory.
- If the file does not exist, then the `names` list is created by multiplying the `name` variable by 100. This creates a list of 100 strings, each containing the user's name.
- Then, the `names` list is written to a binary file named 'names.pkl' using the `pickle.dump()` method.

```
else:
    with open('data/names.pkl', 'rb') as f:
        names=pickle.load(f)
    names=names+[name]*100
    with open('data/names.pkl', 'wb') as f:
        pickle.dump(names, f)
```

- If the 'names.pkl' file already exists, then it is opened for reading using the `open()` method with the `'rb'` mode.
- The `pickle.load()` method is used to read the contents of the file, which is loaded into the `names` variable.
- Then, the `names` list is updated by appending the `name` variable multiplied by 100 to it.
- Finally, the updated `names` list is written back to the 'names.pkl' file using the `pickle.dump()` method.

```
if 'faces_data.pkl' not in os.listdir('data/'):
    with open('data/faces_data.pkl', 'wb') as f:
        pickle.dump(faces_data, f)
```

- The `if` statement checks if a file named 'faces_data.pkl' exists in the 'data/' directory.
- If the file does not exist, then the `faces_data` array is written to a binary file named 'faces_data.pkl' using the `pickle.dump()` method.

```
else:
    with open('data/faces_data.pkl', 'rb') as f:
        faces=pickle.load(f)

    faces=np.append(faces, faces_data, axis=0)
    with open('data/faces_data.pkl', 'wb') as f:
        pickle.dump(faces, f)
```

- If the 'faces_data.pkl' file already exists, then it is opened for reading using the `open()` method with the `'rb'` mode.
- The `pickle.load()` method is used to read the contents of the file, which is loaded into the `faces` variable.

- Then, the `np.append()` method is used to concatenate the `faces_data` array to the `faces` array along the 0th axis. The resulting array is stored back in the `faces` variable.
- Finally, the updated `faces` array is written back to the 'faces_data.pkl' file using the `pickle.dump()` method.