جامعة تشرين

كلية الهندسة الميكانيكية والكهربائية

قسم هندسة الاتصالات و الالكترونيات

السنة الخامسة

برمجة الشبكات

Second Network Programming Homework

اعداد الطالب :

محمد تيسير الراغب

اشراف

الدكتور المهندس مهند عيسى

العام الدراسي

2024/2025

Question 1: Bank ATM Application with TCP Server/Client and Multi-threading
Project Description:
Build a TCP server and client Bank ATM application using Python. The server should handle multiple client connections simultaneously using multi-threading. The application should allow clients to connect, perform banking operations (such as check balance, deposit, and withdraw), and receive their updated account status upon completion.

Requirements: A. The server should be able to handle multiple client connections concurrently. B. The server should maintain a set of pre-defined bank accounts with balances. C. Each client should connect to the server and authenticate with their account details. D. Clients should be able to perform banking operations: check balance, deposit money, and withdraw money. E. The server should keep track of the account balances for each client. F. At the end of the session, the server should send the final account balance to each client.
Guidelines:
• Use Python's socket module without third-party packages.
• Implement multi-threading to handle multiple client connections concurrently.
• Store the account details and balances on the server side.
Notes:
• Write a brief report describing the design choices you made and any challenges faced during implementation.
• You can choose to create a TCP Server/Client Bank ATM application or any other appropriate application that fulfills all requirements.

Bank ATM Application with TCP Server/Client and Multi-threading Design Choices

TCP Server and Client:

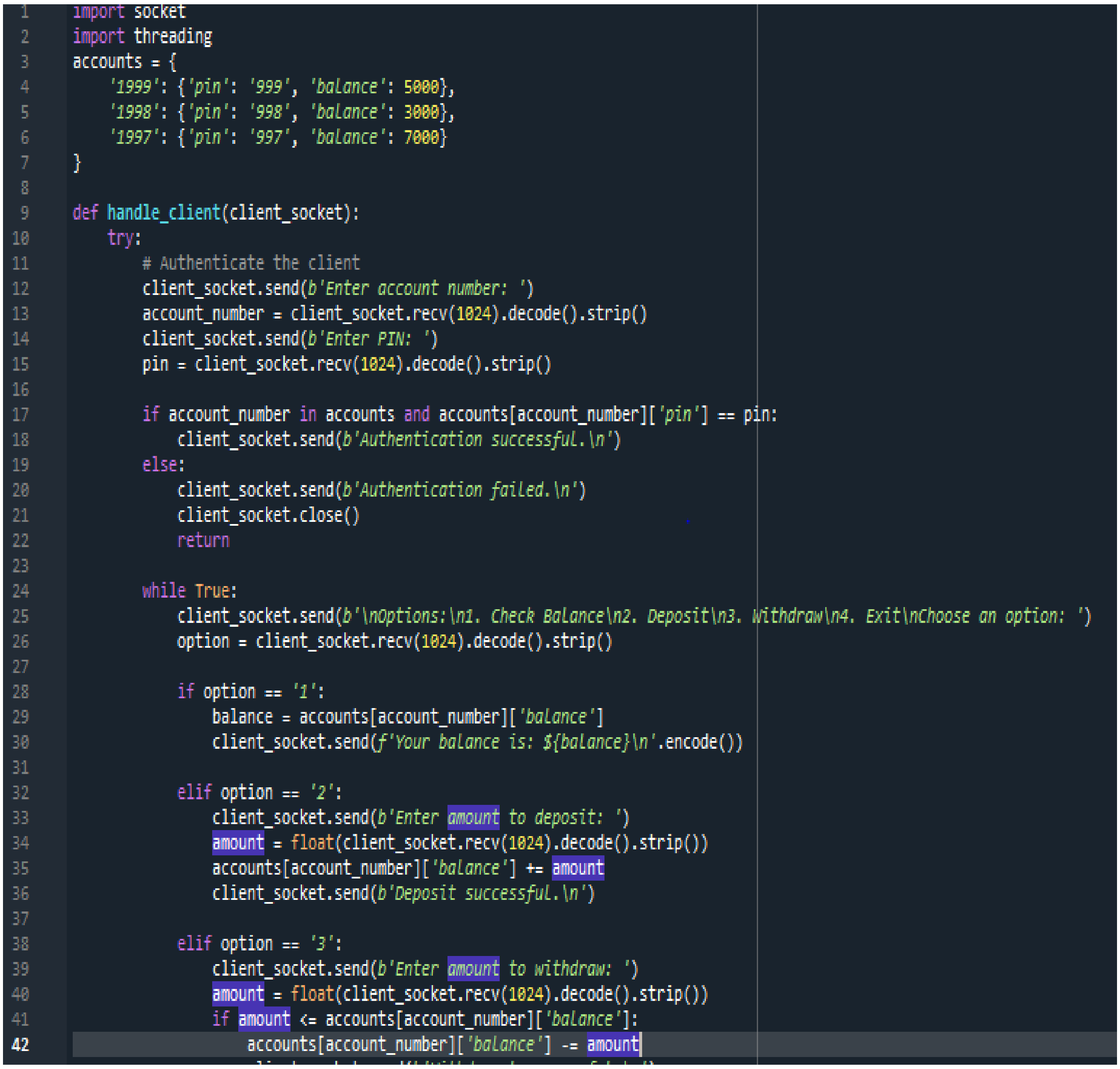
1- Implementing the Bank ATM application using Python's socket module for TCP communication ensures reliable data transfer between the server and clients.
2- **Multi-threading**: Using Python's threading module allows the server to handle multiple client connections concurrently, ensuring responsiveness and scalability.
3- **Pre-defined Accounts**: Storing account details and balances in a dictionary on the server side provides easy access and modification.
4- **Authentication:** Implementing account number and PIN-based authentication ensures secure access to the banking system.
5- **Banking Operations**: Implementing core banking operations (check balance, deposit, withdraw) as server functions ensures central control and consistent handling.

Challenges Faced

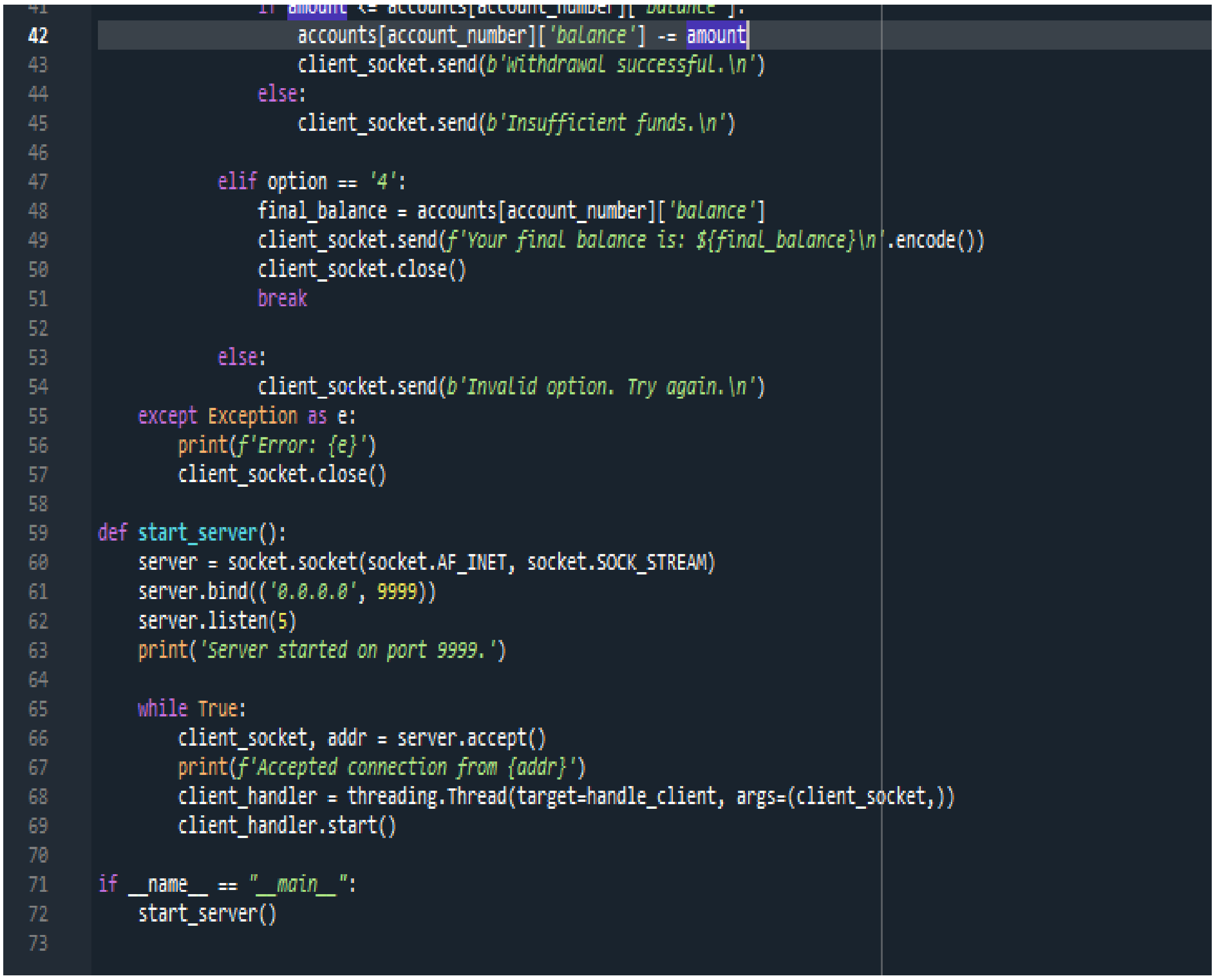
1- **Concurrency Management**: Ensuring thread safety while handling multiple clients required careful synchronization
2- **Error Handling**: Managing various error scenarios (e.g., incorrect input, authentication failures) required robust error handling mechanisms.
3- **Client-Server Communication:** Ensuring smooth communication and proper handling of network interruptions posed challenges.
4- **Data Integrity:** Managing account balances securely and accurately required careful implementation.

Implementation

Here is the implementation of the TCP Server and Client Bank ATM application:

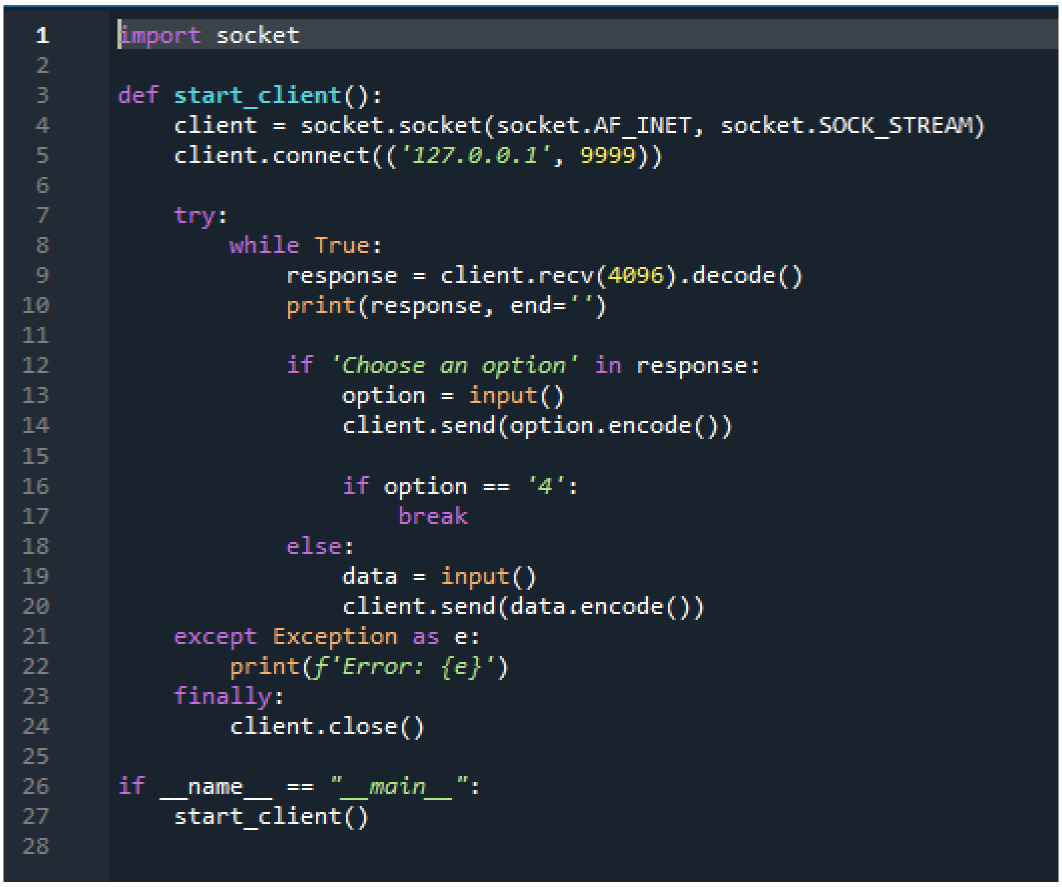


```python
import socket
import threading
accounts = {
    '1999': {'pin': '999', 'balance': 5000},
    '1998': {'pin': '998', 'balance': 3000},
    '1997': {'pin': '997', 'balance': 7000}
}

def handle_client(client_socket):
    try:
        # Authenticate the client
        client_socket.send(b'Enter account number: ')
        account_number = client_socket.recv(1024).decode().strip()
        client_socket.send(b'Enter PIN: ')
        pin = client_socket.recv(1024).decode().strip()

        if account_number in accounts and accounts[account_number]['pin'] == pin:
            client_socket.send(b'Authentication successful.\n')
        else:
            client_socket.send(b'Authentication failed.\n')
            client_socket.close()
            return

        while True:
            client_socket.send(b'\nOptions:\n1. Check Balance\n2. Deposit\n3. Withdraw\n4. Exit\nChoose an option: ')
            option = client_socket.recv(1024).decode().strip()

            if option == '1':
                balance = accounts[account_number]['balance']
                client_socket.send(f'Your balance is: ${balance}\n'.encode())

            elif option == '2':
                client_socket.send(b'Enter amount to deposit: ')
                amount = float(client_socket.recv(1024).decode().strip())
                accounts[account_number]['balance'] += amount
                client_socket.send(b'Deposit successful.\n')

            elif option == '3':
                client_socket.send(b'Enter amount to withdraw: ')
                amount = float(client_socket.recv(1024).decode().strip())
                if amount <= accounts[account_number]['balance']:
                    accounts[account_number]['balance'] -= amount
```



```python
                if amount <= accounts[account_number]['balance']:
                    accounts[account_number]['balance'] -= amount
                    client_socket.send(b'Withdrawal successful.\n')
                else:
                    client_socket.send(b'Insufficient funds.\n')

            elif option == '4':
                final_balance = accounts[account_number]['balance']
                client_socket.send(f'Your final balance is: ${final_balance}\n'.encode())
                client_socket.close()
                break

            else:
                client_socket.send(b'Invalid option. Try again.\n')
    except Exception as e:
        print(f'Error: {e}')
        client_socket.close()

def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('0.0.0.0', 9999))
    server.listen(5)
    print('Server started on port 9999.')

    while True:
        client_socket, addr = server.accept()
        print(f'Accepted connection from {addr}')
        client_handler = threading.Thread(target=handle_client, args=(client_socket,))
        client_handler.start()

if __name__ == "__main__":
    start_server()

```

Client code



```python
import socket

def start_client():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 9999))

    try:
        while True:
            response = client.recv(4096).decode()
            print(response, end='')

            if 'Choose an option' in response:
                option = input()
                client.send(option.encode())

                if option == '4':
                    break
            else:
                data = input()
                client.send(data.encode())
    except Exception as e:
        print(f'Error: {e}')
    finally:
        client.close()

if __name__ == "__main__":
    start_client()

```

Brief Report

Design Choices

1- Multi-threading: We chose multi-threading to handle multiple clients simultaneously. Each client connection runs in a separate thread, allowing the server to manage multiple clients concurrently without blocking.

2- Socket Communication: We used Python's socket module to create a reliable TCP connection between the server and clients.

3- Data Storage: Account details and balances are stored in a dictionary on the server side, enabling quick look-up and update operations.

4- Authentication: Clients are required to authenticate using an account number and PIN, ensuring secure access to their accounts.

5- Banking Operations: The server provides functions for checking balance, depositing, and withdrawing money, ensuring central control over account transactions.

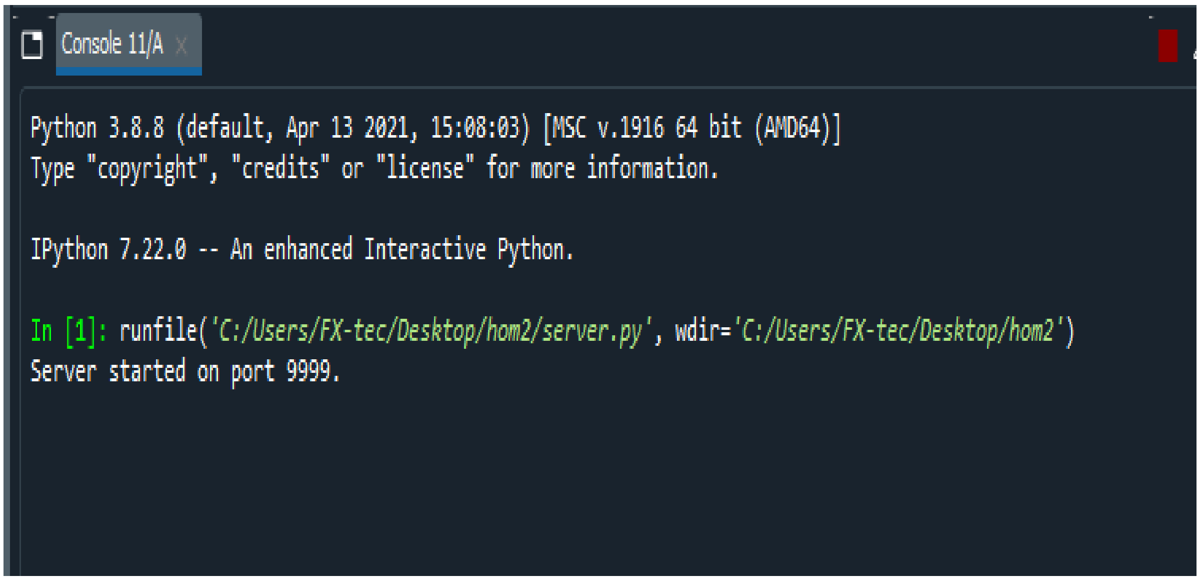
Challenges Faced

1- Concurrency Management: Implementing multi-threading to handle multiple clients concurrently required careful synchronization to avoid data corruption and ensure thread safety.
2- Error Handling: Managing various error scenarios such as incorrect input, authentication failures, and connection issues required robust error handling mechanisms to maintain the stability and reliability of the application.
3- Client-Server Communication: Ensuring smooth and efficient communication between the server and clients, including timely responses to client requests and proper handling of network interruptions, posed challenges during implementation.
4- Data Integrity: Managing account balances securely and accurately required careful implementation to prevent unauthorized access and data loss.

Conclusion

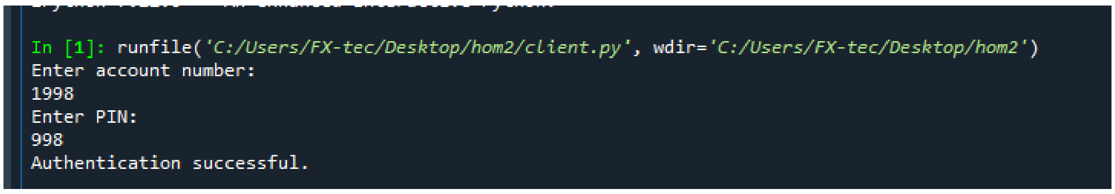
The Bank ATM application with TCP Server/Client and Multi-threading provides a reliable and efficient platform for clients to perform banking operations securely over the network. By leveraging Python's socket module and threading capabilities, we were able to create a scalable and robust solution that meets the requirements of handling multiple client connections concurrently while maintaining accurate account balances and ensuring secure authentication and communication.

١- تشغيل السيرفر



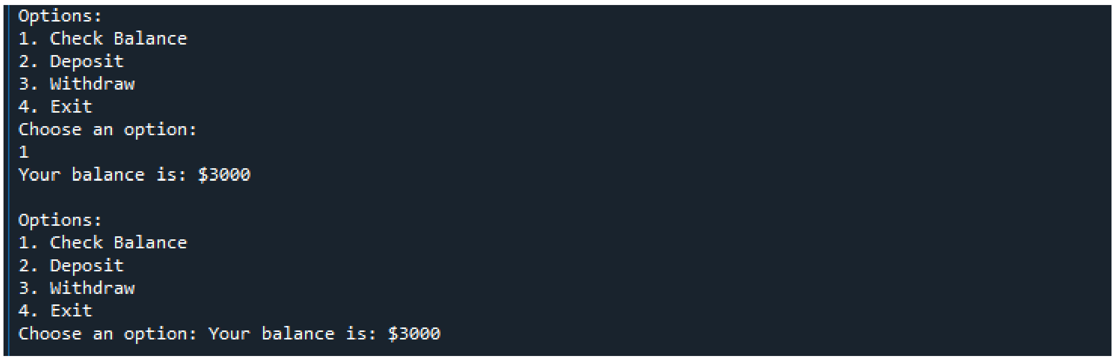
```
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.22.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/FX-tec/Desktop/hom2/server.py', wdir='C:/Users/FX-tec/Desktop/hom2')
Server started on port 9999.
```

٢ – ادخال معلومات الزبون الصحية



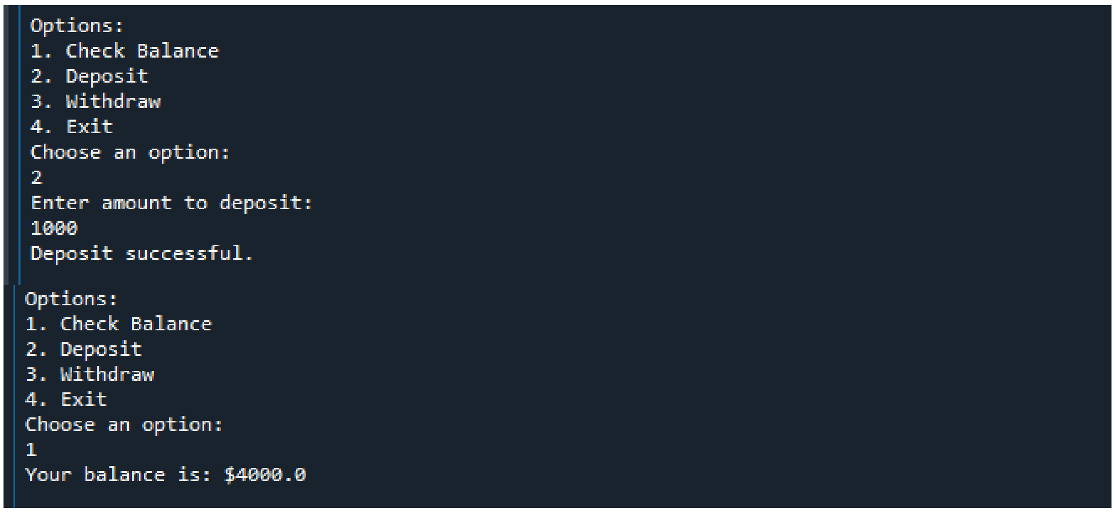
```
In [1]: runfile('C:/Users/FX-tec/Desktop/hom2/client.py', wdir='C:/Users/FX-tec/Desktop/hom2')
Enter account number:
1998
Enter PIN:
998
Authentication successful.
```

٣- القائمة المتضمنة الاجراءات على الحساب (قيمة الحساب الجاري – ايداع – سحب – خروج)



```
Options:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose an option:
1
Your balance is: $3000

Options:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose an option: Your balance is: $3000
```

٥- تم ايداع مبلغ ١٠٠٠ واضافة المبلغ على الحساب

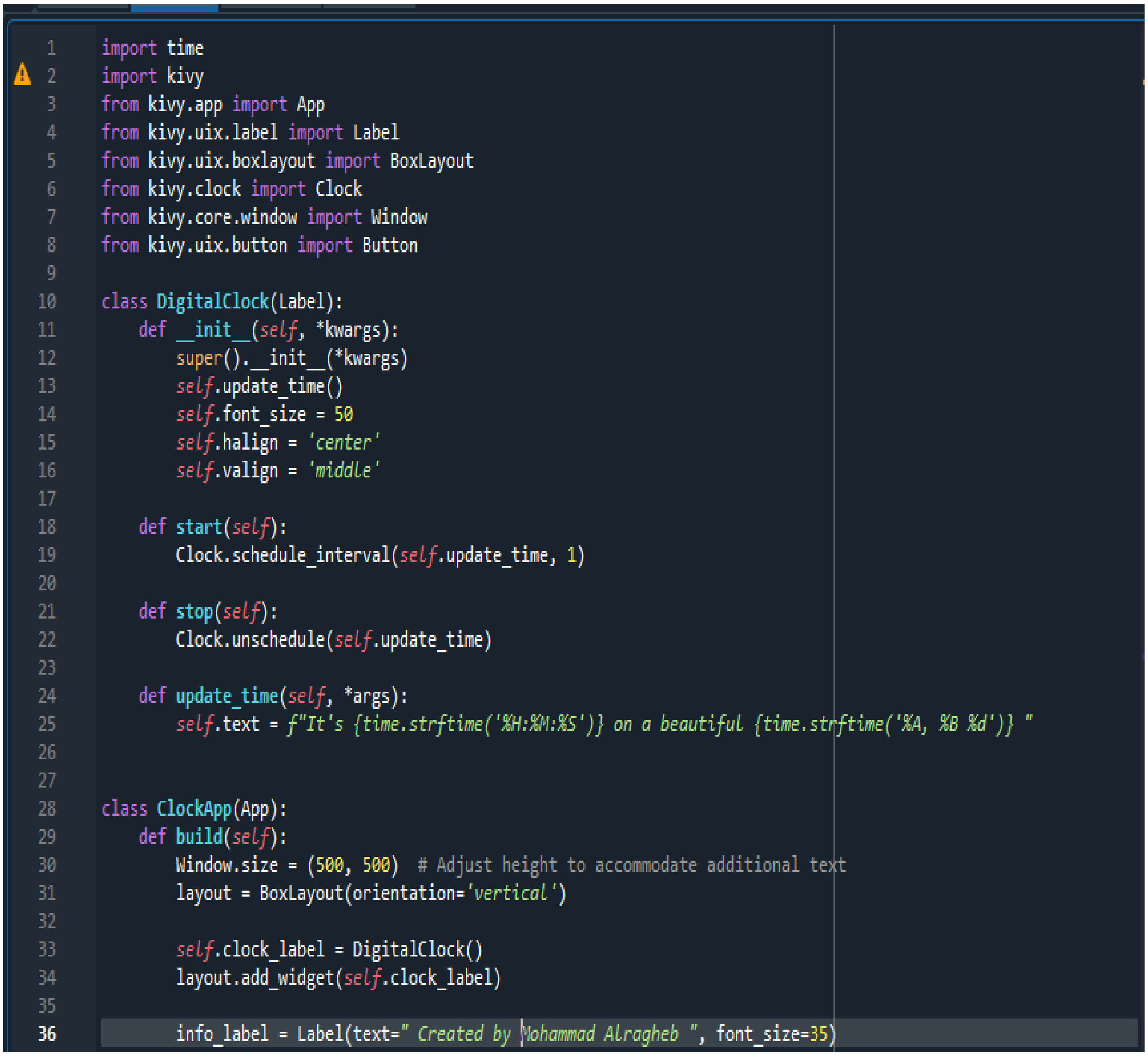


```
Options:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose an option:
2
Enter amount to deposit:
1000
Deposit successful.

Options:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose an option:
1
Your balance is: $4000.0
```

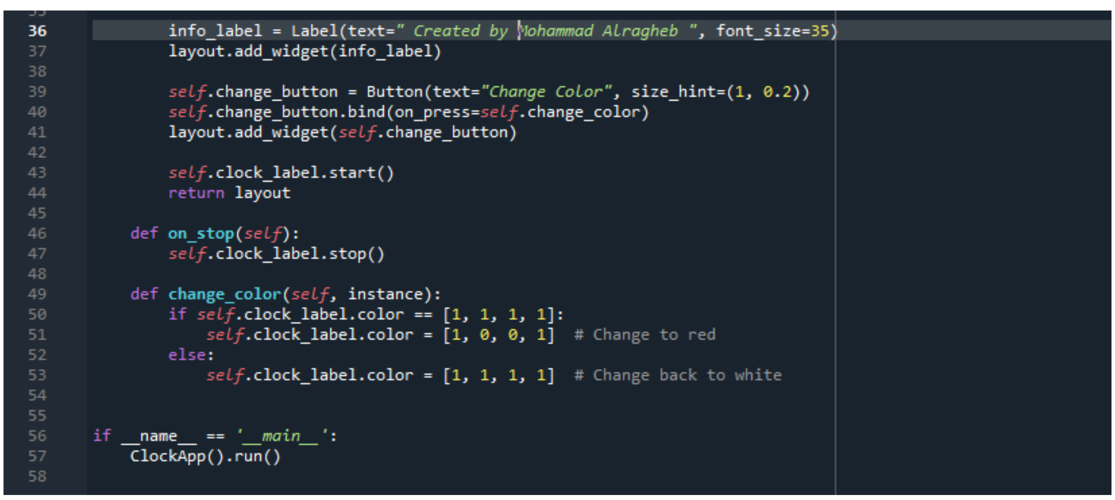
Question 2: Simple Website Project with Python Flask Framework (you have choice to use Django or any Other Deferent Useful Python Project “from provide Project Links”)

انشاء ساعة رقمية

الكود :

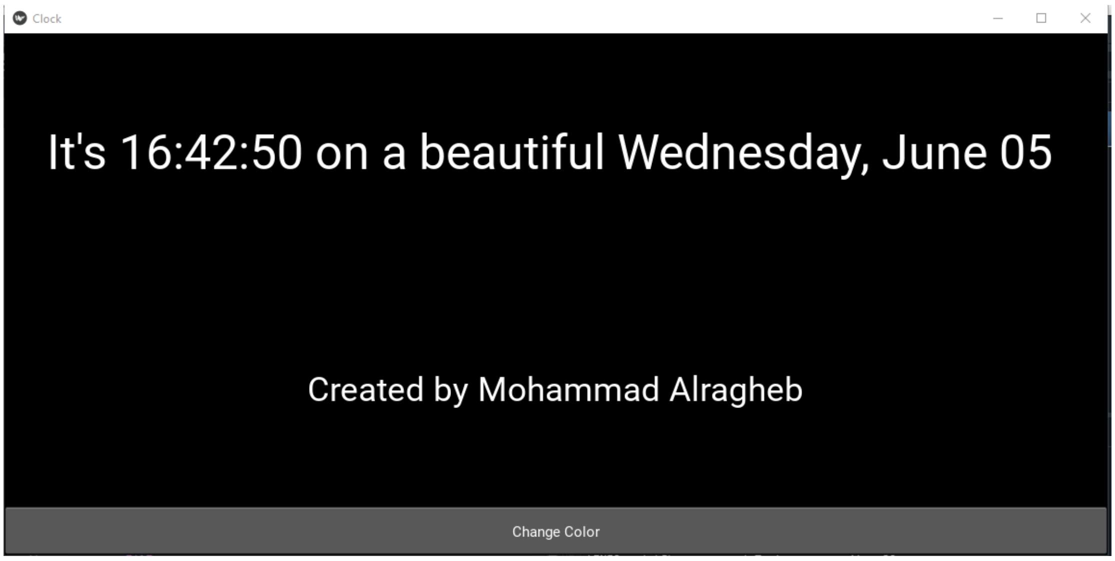


```python
import time
import kivy
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout
from kivy.clock import Clock
from kivy.core.window import Window
from kivy.uix.button import Button

class DigitalClock(Label):
    def __init__(self, *kwargs):
        super().__init__(*kwargs)
        self.update_time()
        self.font_size = 50
        self.halign = 'center'
        self.valign = 'middle'

    def start(self):
        Clock.schedule_interval(self.update_time, 1)

    def stop(self):
        Clock.unschedule(self.update_time)

    def update_time(self, *args):
        self.text = f"It's {time.strftime('%H:%M:%S')} on a beautiful {time.strftime('%A, %B %d')} "


class ClockApp(App):
    def build(self):
        Window.size = (500, 500)  # Adjust height to accommodate additional text
        layout = BoxLayout(orientation='vertical')

        self.clock_label = DigitalClock()
        layout.add_widget(self.clock_label)

        info_label = Label(text=" Created by Mohammad Alragheb ", font_size=35)
```

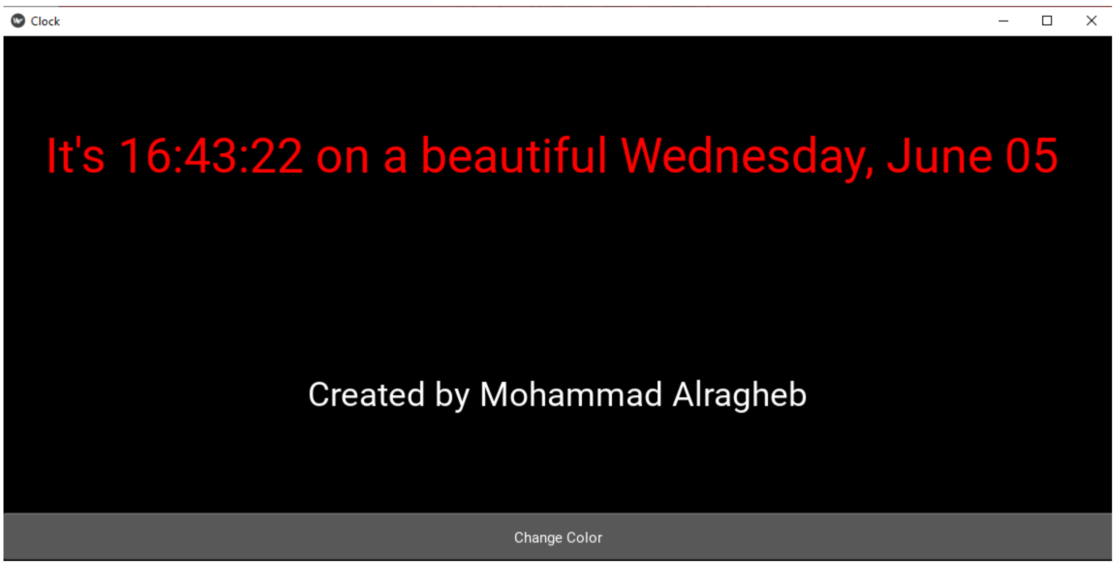


```python
        info_label = Label(text=" Created by Mohammad Alragheb ", font_size=35)
        layout.add_widget(info_label)

        self.change_button = Button(text="Change Color", size_hint=(1, 0.2))
        self.change_button.bind(on_press=self.change_color)
        layout.add_widget(self.change_button)

        self.clock_label.start()
        return layout

    def on_stop(self):
        self.clock_label.stop()

    def change_color(self, instance):
        if self.clock_label.color == [1, 1, 1, 1]:
            self.clock_label.color = [1, 0, 0, 1]  # Change to red
        else:
            self.clock_label.color = [1, 1, 1, 1]  # Change back to white


if __name__ == '__main__':
    ClockApp().run()
```

الخرج



```
Clock

It's 16:42:50 on a beautiful Wednesday, June 05

Created by Mohammad Alragheb

Change Color
```



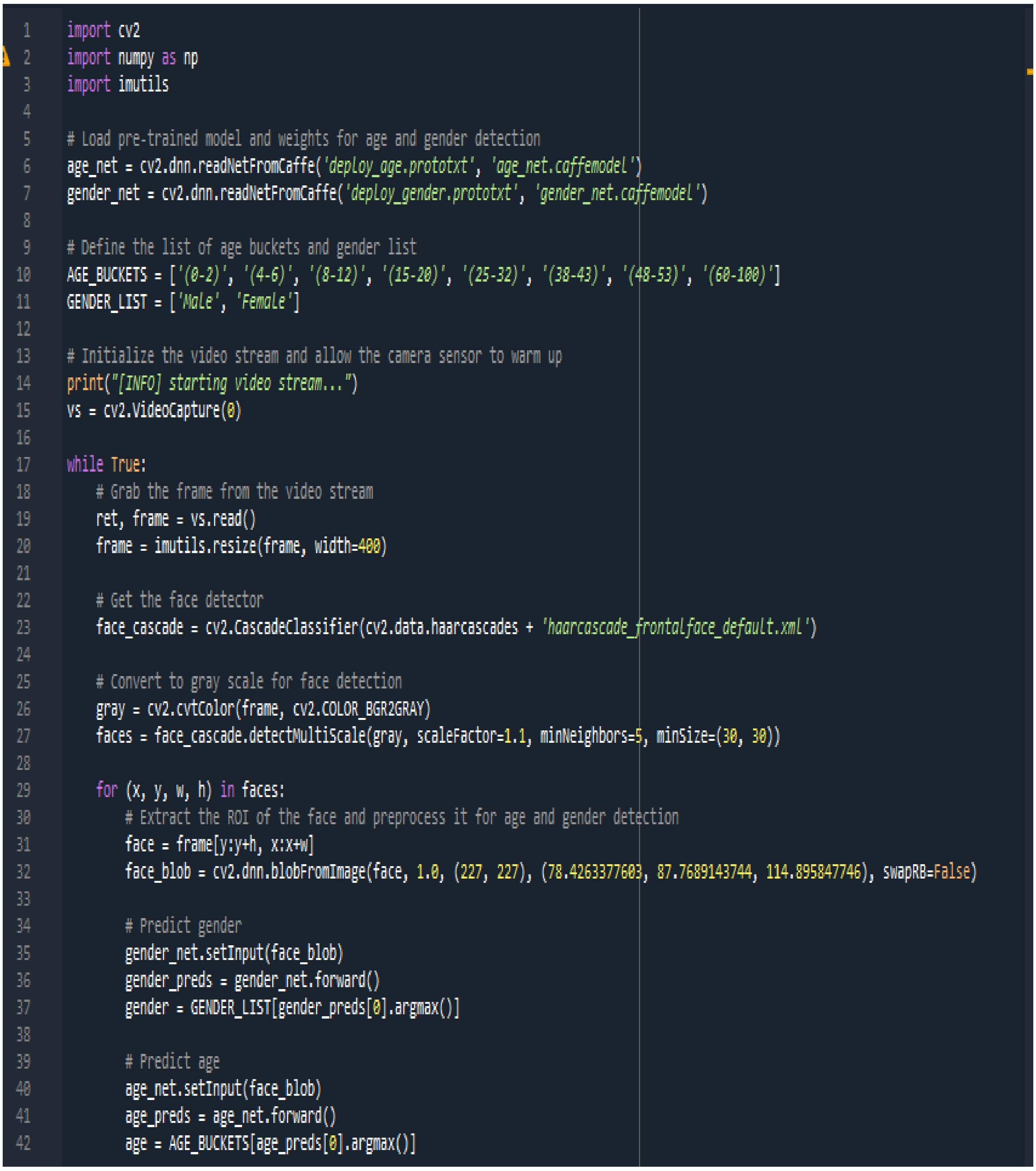
It's 16:43:22 on a beautiful Wednesday, June 05

Created by Mohammad Alragheb

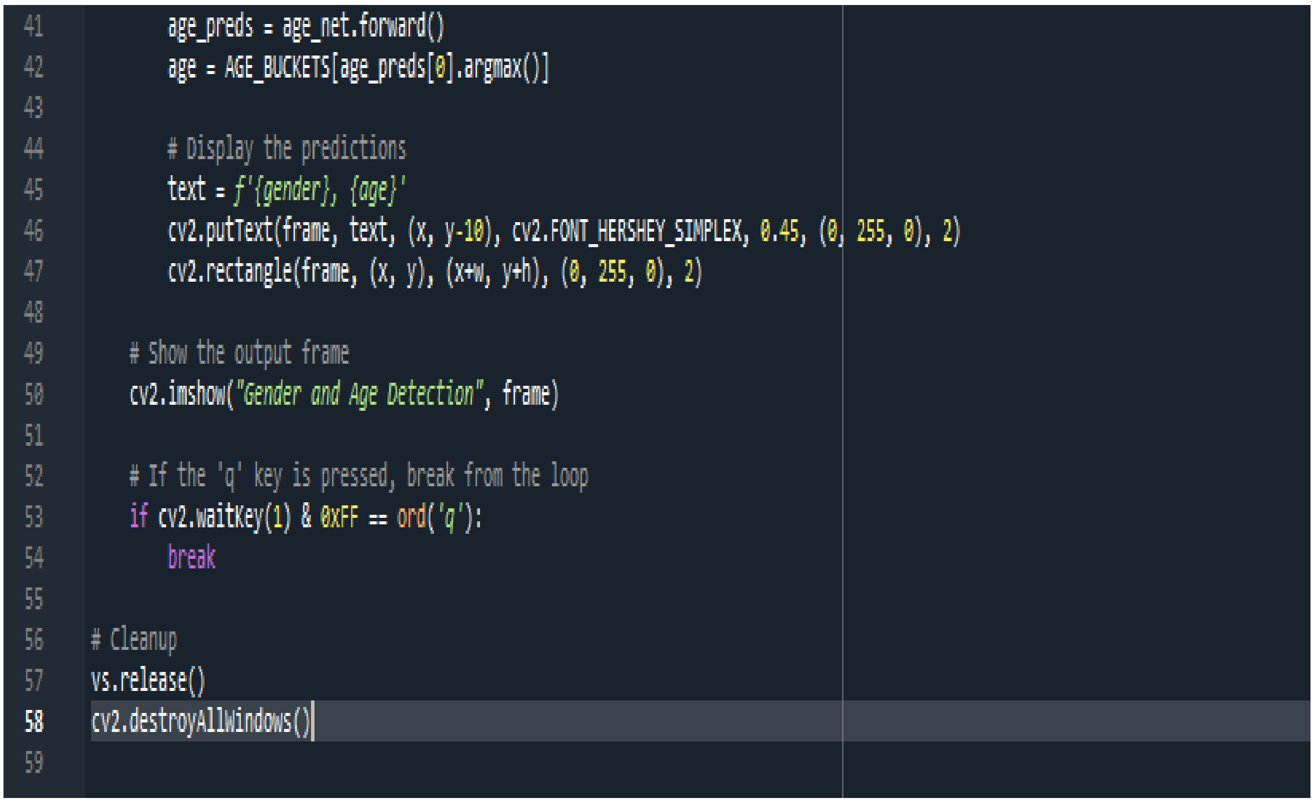
Question 2: Simple Website Project with Python Flask Framework (you have choice to use Django or any Other Deferent Useful Python Project “from provide Project Links”)

Interesting Python Project of Gender and Age Detection with opencv

الكود :

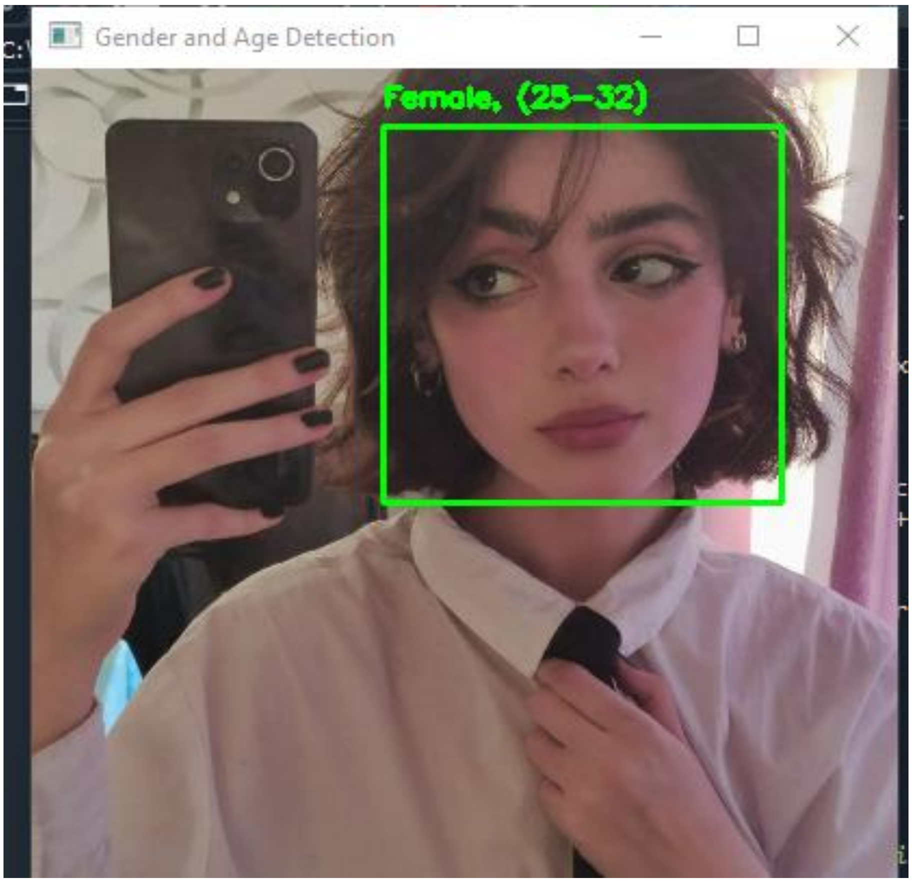


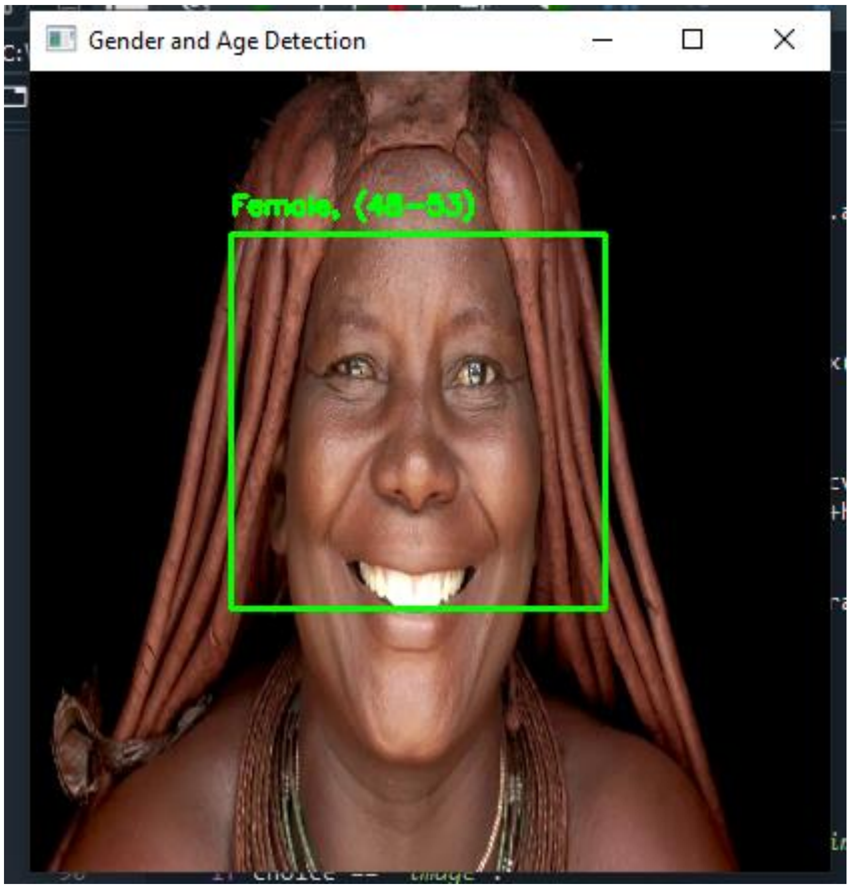
```python
import cv2
import numpy as np
import imutils

# Load pre-trained model and weights for age and gender detection
age_net = cv2.dnn.readNetFromCaffe('deploy_age.prototxt', 'age_net.caffemodel')
gender_net = cv2.dnn.readNetFromCaffe('deploy_gender.prototxt', 'gender_net.caffemodel')

# Define the list of age buckets and gender list
AGE_BUCKETS = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)', '(48-53)', '(60-100)']
GENDER_LIST = ['Male', 'Female']

# Initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = cv2.VideoCapture(0)

while True:
    # Grab the frame from the video stream
    ret, frame = vs.read()
    frame = imutils.resize(frame, width=400)

    # Get the face detector
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

    # Convert to gray scale for face detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        # Extract the ROI of the face and preprocess it for age and gender detection
        face = frame[y:y+h, x:x+w]
        face_blob = cv2.dnn.blobFromImage(face, 1.0, (227, 227), (78.4263377603, 87.7689143744, 114.895847746), swapRB=False)

        # Predict gender
        gender_net.setInput(face_blob)
        gender_preds = gender_net.forward()
        gender = GENDER_LIST[gender_preds[0].argmax()]

        # Predict age
        age_net.setInput(face_blob)
        age_preds = age_net.forward()
        age = AGE_BUCKETS[age_preds[0].argmax()]
```



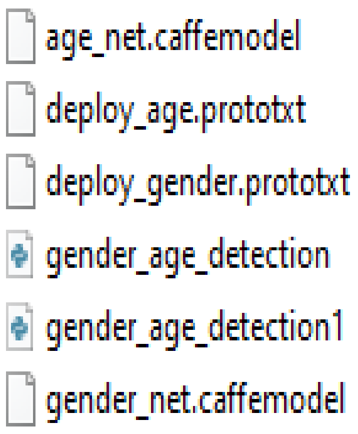
```python
        age_preds = age_net.forward()
        age = AGE_BUCKETS[age_preds[0].argmax()]

        # Display the predictions
        text = f'{gender}, {age}'
        cv2.putText(frame, text, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Show the output frame
    cv2.imshow("Gender and Age Detection", frame)

    # If the 'q' key is pressed, break from the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Cleanup
vs.release()
cv2.destroyAllWindows()
```

الخرج :





الملفات الخارجية المستخدمة من اجل التعرف على العمر والجنس



age_net.caffemodel
deploy_age.prototxt
deploy_gender.prototxt
gender_age_detection
gender_age_detection1
gender_net.caffemodel