Name: Sahil Thakur **Sem:** VII

PRN: 21070521065 **Sec:** B(1)

GenAI CA – II (Assignment)

Q:1 Generate a model in Python for representation of a bank account of type savings and balance along with transactions of deposit and withdrawals and currently create a program to generate 100 accounts with Random balance and transactions for no. of months and no. of transactions with a seed value of amount. Print all 100 accounts with the last balance and organize them by lowest to highest balance.

I have created a `BankAccount` class to represent a savings account with basic functionalities like depositing and withdrawing money. For each transaction, the account balance is updated, and the transaction details are recorded.

I then defined functions to:

1. Generate random transactions for each account (`generate_transactions`).
2. Create 100 bank accounts with random initial balances and simulate random transactions over a period of 12 months (`generate_accounts`).
3. Sort and print the accounts based on their final balance (`print_accounts_sorted`).

Randomization ensures that each account has different transaction histories and balances.

Brief explanation of each function and its purpose:

1. **`deposit(self, amount)`**: Adds the specified amount to the account balance and logs the transaction.
2. **`withdraw(self, amount)`**: Subtracts the specified amount from the account balance if sufficient funds are available; otherwise, logs a failed transaction.
3. **`generate_transactions(account, num_months, num_transactions_per_month)`**: Simulates random transactions (either deposits or withdrawals) for the given account over several months, with a specified number of transactions per month.
4. **`generate_accounts(num_accounts, num_months, num_transactions_per_month, seed)`**: Creates a specified number of bank accounts, each with a random initial balance.
5. **`print_accounts_sorted(accounts)`**: Sorts the accounts by their final balance and prints the account details in ascending order.

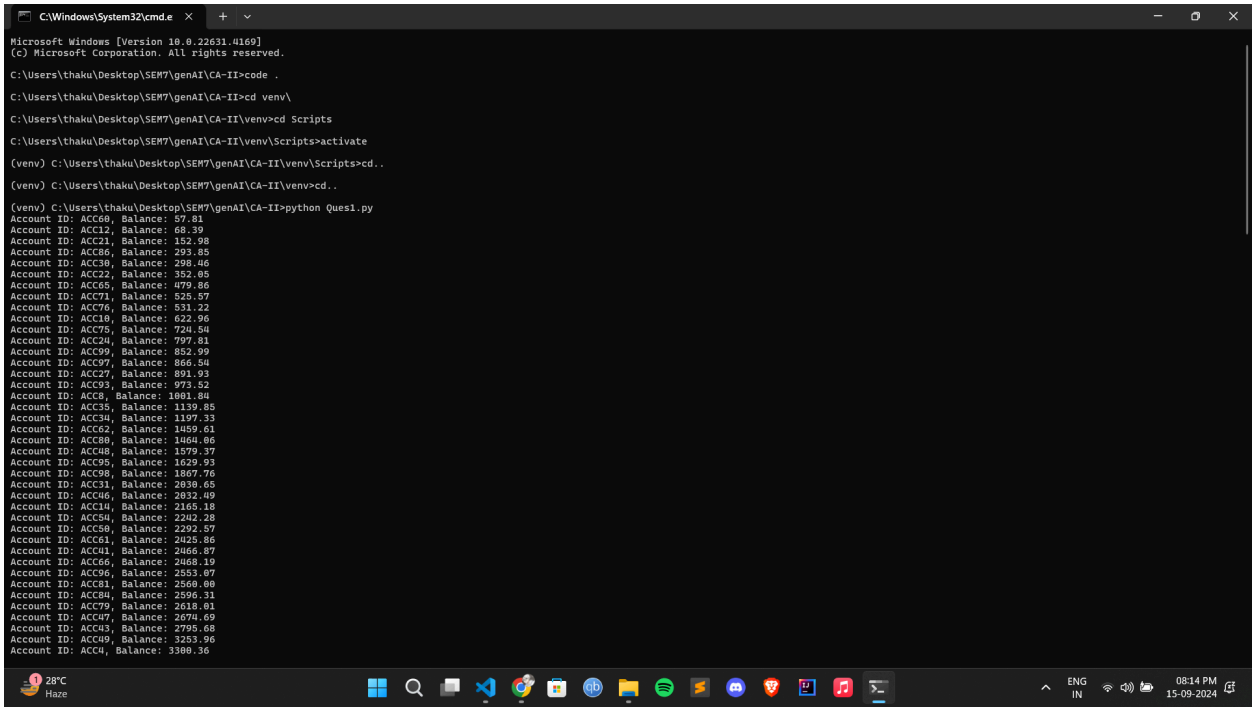
Code:

```python
"""Q:1 Generate a model in Python for representation of a bank account of type savings and
balance along with transactions of deposit and withdrawals and currently create a program to
generate 100 accounts with Random balance and transactions for no. of months and no. of
transactions with a seed value of amount. Print all 100 accounts with the last balance and
organize them by lowest to highest balance."""


import random

# Define a BankAccount class for Savings account
class BankAccount:
    def __init__(self, account_id, initial_balance):
        self.account_id = account_id
        self.balance = initial_balance
        self.transactions = []

    def deposit(self, amount):
        self.balance += amount
        self.transactions.append(f"Deposit: +{amount}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            self.transactions.append(f"Withdraw: -{amount}")
        else:
            self.transactions.append(f"Failed Withdraw (Insufficient funds): -{amount}")

    def __repr__(self):
        return f"Account ID: {self.account_id}, Balance: {self.balance:.2f}"



# Function to create random transactions for a given account
def generate_transactions(account, num_months, num_transactions_per_month):
    for _ in range(num_months):
        for _ in range(num_transactions_per_month):
            transaction_type = random.choice(['deposit', 'withdraw'])
```

```python
# Function to create random transactions for a given account
def generate_transactions(account, num_months, num_transactions_per_month):
    for _ in range(num_months):
        for _ in range(num_transactions_per_month):
            transaction_type = random.choice(['deposit', 'withdraw'])
            amount = random.uniform(100, 1000)
            if transaction_type == 'deposit':
                account.deposit(amount)
            else:
                account.withdraw(amount)



# Function to generate 100 random bank accounts
def generate_accounts(num_accounts, num_months, num_transactions_per_month, seed):
    accounts = []
    random.seed(seed)

    for i in range(num_accounts):
        account_seed = seed + i
        random.seed(account_seed)

        initial_balance = random.uniform(1000, 5000)
        account = BankAccount(account_id=f"ACC{i+1}", initial_balance=initial_balance)

        generate_transactions(account, num_months, num_transactions_per_month)
        accounts.append(account)

    return accounts


def print_accounts_sorted(accounts):
    sorted_accounts = sorted(accounts, key=lambda x: x.balance)
    for account in sorted_accounts:
        print(account)
```

```python
if __name__ == "__main__":
    seed_value = 42
    num_months = 12
    num_transactions_per_month = 5

    accounts = generate_accounts(num_accounts=100, num_months=num_months, num_transactions_per_month=num_transactions_per_month, seed=seed_value)
    print_accounts_sorted(accounts)
```

Output:



```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4169]
(c) Microsoft Corporation. All rights reserved.

C:\Users\thaku\Desktop\SEM7\genAI\CA-II>code .

C:\Users\thaku\Desktop\SEM7\genAI\CA-II>cd venv\

C:\Users\thaku\Desktop\SEM7\genAI\CA-II\venv>cd Scripts

C:\Users\thaku\Desktop\SEM7\genAI\CA-II\venv\Scripts>activate

(venv) C:\Users\thaku\Desktop\SEM7\genAI\CA-II\venv\Scripts>cd..

(venv) C:\Users\thaku\Desktop\SEM7\genAI\CA-II\venv>cd..

(venv) C:\Users\thaku\Desktop\SEM7\genAI\CA-II>python Ques1.py
Account ID: ACC60, Balance: 57.81
Account ID: ACC12, Balance: 68.39
Account ID: ACC21, Balance: 152.98
Account ID: ACC86, Balance: 293.85
Account ID: ACC30, Balance: 298.46
Account ID: ACC22, Balance: 352.05
Account ID: ACC65, Balance: 479.86
Account ID: ACC71, Balance: 525.57
Account ID: ACC76, Balance: 531.22
Account ID: ACC10, Balance: 622.96
Account ID: ACC75, Balance: 724.54
Account ID: ACC24, Balance: 797.81
Account ID: ACC99, Balance: 852.99
Account ID: ACC97, Balance: 866.54
Account ID: ACC27, Balance: 891.93
Account ID: ACC93, Balance: 973.52
Account ID: ACC8, Balance: 1001.84
Account ID: ACC35, Balance: 1139.85
Account ID: ACC34, Balance: 1197.33
Account ID: ACC62, Balance: 1459.61
Account ID: ACC80, Balance: 1464.06
Account ID: ACC48, Balance: 1579.37
Account ID: ACC95, Balance: 1629.93
Account ID: ACC98, Balance: 1867.76
Account ID: ACC31, Balance: 2030.65
Account ID: ACC46, Balance: 2032.49
Account ID: ACC14, Balance: 2165.18
Account ID: ACC54, Balance: 2242.28
Account ID: ACC50, Balance: 2292.57
Account ID: ACC61, Balance: 2425.86
Account ID: ACC41, Balance: 2466.87
Account ID: ACC66, Balance: 2468.19
Account ID: ACC96, Balance: 2553.07
Account ID: ACC81, Balance: 2560.00
Account ID: ACC84, Balance: 2596.31
Account ID: ACC79, Balance: 2618.01
Account ID: ACC47, Balance: 2674.69
Account ID: ACC43, Balance: 2795.68
Account ID: ACC49, Balance: 3253.96
Account ID: ACC4, Balance: 3300.36
```

Q:6 Generate a model to represent a mathematical equation, write a program to parse the equation, and ask for input for each parameter.

My Python program is aimed at modeling and solving mathematical equations by entering the equation and values of its parameters. It relies on the Sympy which is a very useful library for symbolic computations in python.

The program starts by the import of the required symbolic expression controlling library known as SymPy. It then goes on to define the parse_equation function with the task of converting an equation provided by the user in string form to the symbolic equation format using sympify(). This makes it possible for the program to manipulate and solve the equation in a form that is not static but in the form of a string.

After, the ask_for_inputs function checks the variables required in the equation and asks the user to insert values for the said variables. The variables required are obtained using SymPy's free_symbols method; all the inputs given by the user are stored in a dictionary format, where the key represents the variable, while the value represents the value entered by the user.

The solve_equation function accepts the symbolic equation and replaces the values of the variables that have been input by the user using the subs() method. It means that at the end of the day it is possible to evaluate the equation with the help of certain inputs.

Code:

```python
"""Q:6 Generate a model to represent a mathematical equation, write a program to parse the
equation, and ask for input for each parameter."""


import sympy as sp

def parse_equation(equation_str):
    """
    Parse a given mathematical equation string into a symbolic equation.
    """
    equation = sp.sympify(equation_str)
    return equation



def ask_for_inputs(variables):
    """
    Ask the user for input values for each variable in the equation.
    """
    inputs = {}
    for var in variables:
        value = float(input(f"Enter the value for {var}: "))
        inputs[var] = value
    return inputs


def solve_equation(equation, inputs):
    """
    Substitute the inputs into the equation and solve it.

    """
    result = equation.subs(inputs)
    return result

def main():
    equation_str = input("Enter the mathematical equation (e.g., 2*x + 3*y - z): ")
    equation = parse_equation(equation_str)
```

```python
    return result

def main():
    equation_str = input("Enter the mathematical equation (e.g., 2*x + 3*y - z): ")
    equation = parse_equation(equation_str)

    variables = equation.free_symbols
    inputs = ask_for_inputs(variables)

    result = solve_equation(equation, inputs)
    print(f"The result of the equation is: {result}")


if __name__ == "__main__":
    main()
```

Output:

```
(venv) C:\Users\thaku\Desktop\SEM7\genAI\CA-II>python Ques6.py
Enter the mathematical equation (e.g., 2*x + 3*y - z): 2*x + 3*y - z
Enter the value for y: 1
Enter the value for z: 4
Enter the value for x: 10
The result of the equation is: 19.0000000000000

(venv) C:\Users\thaku\Desktop\SEM7\genAI\CA-II>
```