CA-2

Priyanshu Bawse

21070521054

Q1. 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


class SavingsAccount:
    def __init__(self, account_number, initial_balance=0):
        self.account_number = account_number
        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"Withdraw failed (Insufficient funds): -{amount}")

    def __repr__(self):
        return f"Account {self.account_number} - Balance: {self.balance}"

def generate_random_transactions(account, num_months, num_transactions_per_month, seed_value):
    random.seed(seed_value)
    for _ in range(num_months):
        for _ in range(num_transactions_per_month):
            transaction_type = random.choice(['deposit', 'withdraw'])
            amount = random.randint(1, 1000)

            if transaction_type == 'deposit':
                account.deposit(amount)
            else:
                account.withdraw(amount)

def generate_accounts(num_accounts, num_months, num_transactions, seed_value):
    accounts = []
    for i in range(1, num_accounts + 1):
        initial_balance = random.randint(1000, 10000)
```

```
        account = SavingsAccount(account_number=i,
initial_balance=initial_balance)
        generate_random_transactions(account, num_months, num_transactions,
seed_value)
        accounts.append(account)


    accounts.sort(key=lambda acc: acc.balance)


    return accounts


NUM_ACCOUNTS = 100
NUM_MONTHS = 12
NUM_TRANSACTIONS_PER_MONTH = 10
SEED_VALUE = 42


accounts = generate_accounts(NUM_ACCOUNTS, NUM_MONTHS,
NUM_TRANSACTIONS_PER_MONTH, SEED_VALUE)


for account in accounts:
    print(account)
```

Q2. Generate a model to represent a mathematical equation, write a program to parse the equation, and ask for input for each parameter

```
import random


class SavingsAccount:
    def __init__(self, account_number, initial_balance=0):
```

```
        self.account_number = account_number
        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"Withdraw failed (Insufficient funds): -
{amount}")

    def __repr__(self):
        return f"Account {self.account_number} - Balance: {self.balance}"

def generate_random_transactions(account, num_months,
num_transactions_per_month, seed_value):
    random.seed(seed_value)
    for _ in range(num_months):
        for _ in range(num_transactions_per_month):
            transaction_type = random.choice(['deposit', 'withdraw'])
            amount = random.randint(1, 1000)
```

```
            if transaction_type == 'deposit':
                account.deposit(amount)
            else:
                account.withdraw(amount)


def generate_accounts(num_accounts, num_months, num_transactions,
seed_value):
    accounts = []
    for i in range(1, num_accounts + 1):
        initial_balance = random.randint(1000, 10000)
        account = SavingsAccount(account_number=i,
initial_balance=initial_balance)
        generate_random_transactions(account, num_months, num_transactions,
seed_value)
        accounts.append(account)


    accounts.sort(key=lambda acc: acc.balance)


    return accounts


NUM_ACCOUNTS = 100
NUM_MONTHS = 12
NUM_TRANSACTIONS_PER_MONTH = 10
SEED_VALUE = 42


accounts = generate_accounts(NUM_ACCOUNTS, NUM_MONTHS,
NUM_TRANSACTIONS_PER_MONTH, SEED_VALUE)
```

```
for account in accounts:
    print(account)
```