PRN:21070521035

NAME: Kashish Sharma

SUBJECT: GENAI CA2 submission

**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.

**Solution-🡪**

#PRN:21070521035 NAME:Kashish Sharma

import random

class BankAccount:

    def \_\_init\_\_(self, account\_id, balance=0):

        self.account\_id = account\_id

        self.balance = balance

        self.transactions = []

    def deposit(self, amount):

        self.balance += amount

        self.transactions.append(f"Deposit: {amount}")

    def withdraw(self, amount):

        if self.balance >= amount:

            self.balance -= amount

            self.transactions.append(f"Withdrawal: {amount}")

        else:

            self.transactions.append(f"Withdrawal Failed: Insufficient funds")

    def \_\_repr\_\_(self):

        return f"Account ID: {self.account\_id}, Final Balance: {self.balance}, Transactions: {len(self.transactions)}"

def generate\_accounts(num\_accounts, num\_months, seed):

    random.seed(seed)

    accounts = []

    for i in range(num\_accounts):

        initial\_balance = random.randint(1000, 5000)

        account = BankAccount(account\_id=i+1, balance=initial\_balance)

        for month in range(num\_months):

            num\_transactions = random.randint(1, 10)

            for \_ in range(num\_transactions):

                transaction\_type = random.choice(['deposit', 'withdraw'])

                amount = random.randint(100, 1000)

                if transaction\_type == 'deposit':

                    account.deposit(amount)

                elif transaction\_type == 'withdraw':

                    account.withdraw(amount)

        accounts.append(account)

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

    return accounts

num\_accounts = 100

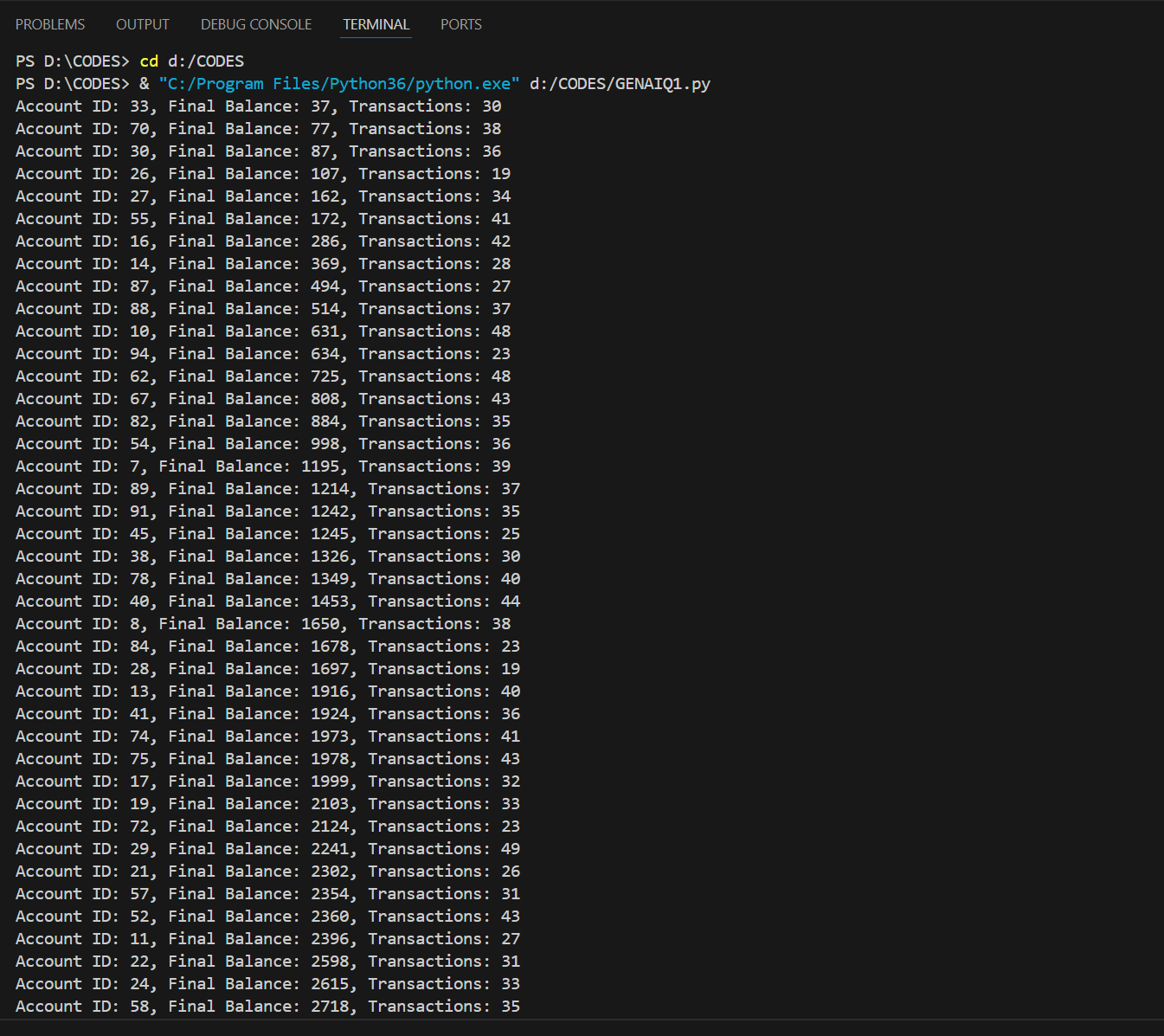
num\_months = 6

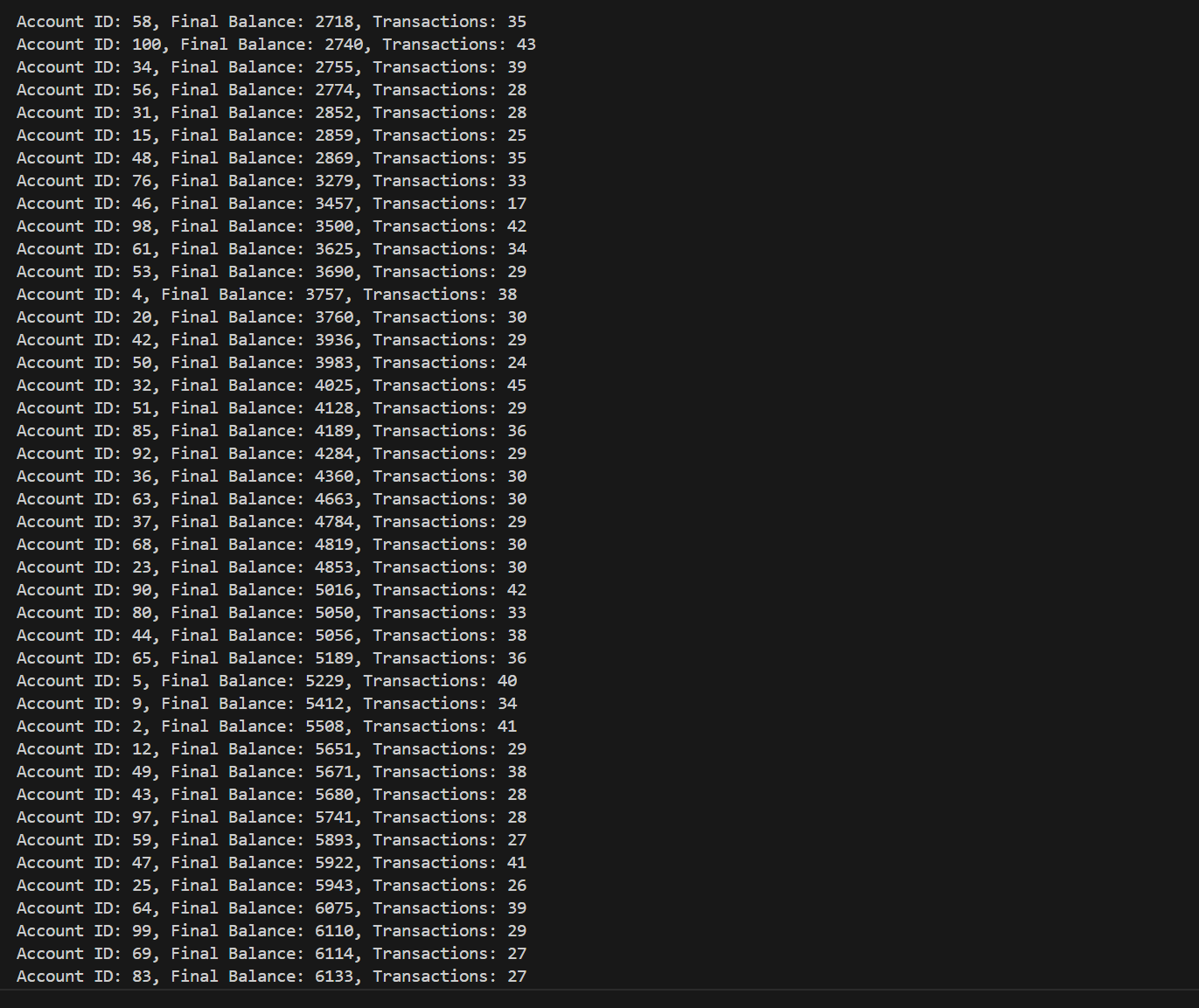
seed\_value = 42

accounts = generate\_accounts(num\_accounts, num\_months, seed\_value)

for account in accounts:

    print(account)





**EXPLAINATION:**

It contains a notebook and a script to imitate the work for 6 months with 100 accounts in the bank. Every test account is to be identified by a specific ID and has an initial randomly set balance. Such random profits and losses are incorporated as a series of transactions to simulate any standard banking process such as deposits and withdrawals.

**Account Initialization:**

Bank Accounts: Creates 100 instances of BankAccount, each have a distinct ID and has an initial balance betwween $1000 and $5000. Each of the accounts also has a different holder name, which is created as ‘Holder\_1’, ‘Holder\_2’ and so on.

**Transaction Simulation:**

Monthly Activity: Thus, for every account on average, the code demonstrates banking transactions during a 6- month cycle. Unlike in each week, a random number of transactions (which is randomly produced in the range of 1-10) occurs in each month.

**Transaction Details**: Deposits and withdrawals are models randomly chosen out of transactions. Each of the transactions can occur in the range for $100 to $1000. The dates of each transaction are also randomised within the month so as to reduce variability in the timing of the transactions.

**Data Processing:**

Balance Calculation: At the end of each simulation period, the accounts balance for the entire period is calculated.

Sorting Accounts: It means that accounts are placed in order according to their outstanding balance starting with the lowest.

**Output:**

A compact overview of the data can be obtained from balance sheets by providing the first ten levied account balances in a descending order; and the last ten levied account balances in ascending order. This summary acts as a brief outlook of how the account is in terms of standing without much detail information.

**Purpose:**

In this simulation, actual figures of accounts of a bank are generated to show their performance with random transactions. It demonstrates account management, record processing and sorting of data which is helpful in understanding balances and account transactions.

**Q:2** Generate a model in Python to represent a Housing loan scheme and create a chart to

display the Emi based on rate of interest and reducing balance for a given period. If a customer

wishes to close the loan earlier, print the interest lost distributed over the remaining no. Of

months. Assume suitable data and inputs as necessary.

**Solution🡪**

#PRN:21070521035 NAME:Kashish Sharma

import matplotlib.pyplot as plt

import numpy as np

class HousingLoan:

    def \_\_init\_\_(self, principal, annual\_rate, months):

        self.principal = principal

        self.annual\_rate = annual\_rate

        self.months = months

        self.monthly\_rate = annual\_rate / 12 / 100

    def calculate\_emi(self):

        emi = self.principal \* self.monthly\_rate \* (1 + self.monthly\_rate)\*\*self.months / ((1 + self.monthly\_rate)\*\*self.months - 1)

        return emi

    def calculate\_amortization\_schedule(self):

        emi = self.calculate\_emi()

        balance = self.principal

        schedule = []

        for \_ in range(self.months):

            interest\_payment = balance \* self.monthly\_rate

            principal\_payment = emi - interest\_payment

            balance -= principal\_payment

            schedule.append((emi, interest\_payment, principal\_payment, balance))

        return schedule

    def plot\_emi\_chart(self):

        schedule = self.calculate\_amortization\_schedule()

        months = np.arange(1, self.months + 1)

        emis = [x[0] for x in schedule]

        balances = [x[3] for x in schedule]

        plt.figure(figsize=(12, 6))

        plt.plot(months, emis, label='EMI', color='blue')

        plt.plot(months, balances, label='Remaining Balance', color='red')

        plt.xlabel('Month')

        plt.ylabel('Amount')

        plt.title('EMI and Remaining Balance Over Time')

        plt.legend()

        plt.grid(True)

        plt.show()

    def calculate\_early\_closure\_interest(self, remaining\_months):

        original\_schedule = self.calculate\_amortization\_schedule()

        remaining\_schedule = original\_schedule[:remaining\_months]

        total\_paid = sum(x[0] for x in remaining\_schedule)

        remaining\_balance = remaining\_schedule[-1][3] if remaining\_months > 0 else 0

        total\_interest\_paid = total\_paid - (self.principal - remaining\_balance)

        interest\_loss\_per\_month = total\_interest\_paid / remaining\_months if remaining\_months > 0 else 0

        return interest\_loss\_per\_month

principal = 500000

annual\_rate = 7.5

months = 240

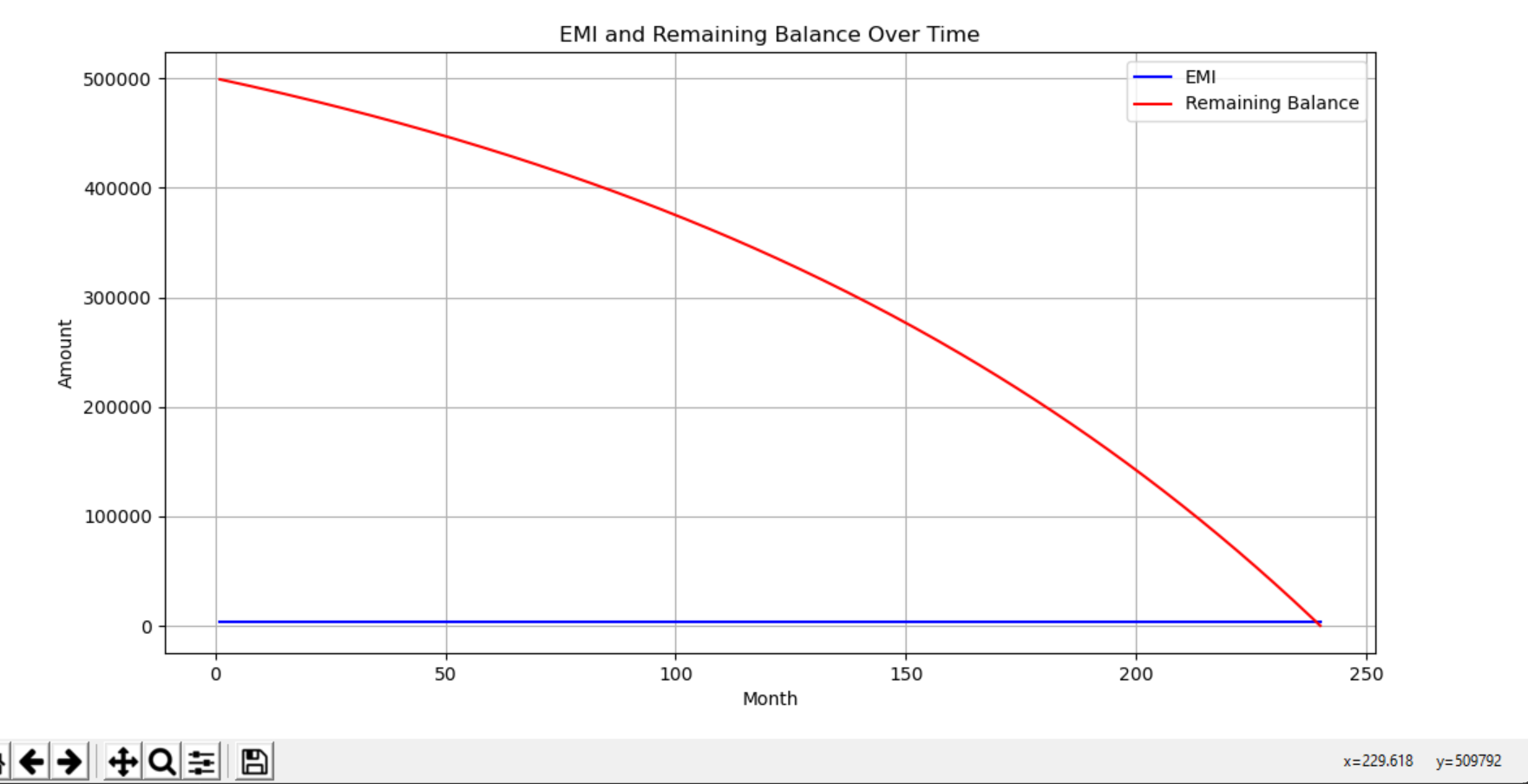
loan = HousingLoan(principal, annual\_rate, months)

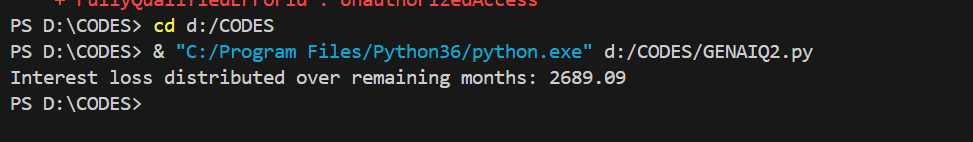
loan.plot\_emi\_chart()

remaining\_months = 120

interest\_loss\_per\_month = loan.calculate\_early\_closure\_interest(remaining\_months)

print(f"Interest loss distributed over remaining months: {interest\_loss\_per\_month:.2f}")

****

****

**EXPLAINATION:**

Defined the HousingLoan Class:

* Created a class to represent a housing loan, encapsulating the necessary attributes and methods for loan calculation and analysis.

Initialized Loan Parameters:

* Set up the loan with a principal amount, annual interest rate, and duration in months.

Calculated EMI:

* Implemented a method to compute the Equated Monthly Installment (EMI) using the formula for a reducing balance loan, providing the fixed monthly payment required over the loan term.

Generated Amortization Schedule:

* Developed a method to create an amortization schedule that tracks:
  + Monthly EMI payments
  + Interest payments
  + Principal repayments
  + Remaining balance
* This schedule helps in understanding how each payment impacts the loan balance and how the composition of payments changes over time.

Visualized Loan Payments:

* Created a chart to visually represent:
  + The EMI payments over time
  + The remaining balance of the loan
* Used matplotlib to plot these data points, providing a clear view of how payments and the remaining balance evolve throughout the loan term.

Calculated Interest Loss for Early Closure:

* Added functionality to calculate the financial impact of closing the loan early, including:
  + The total interest paid up to the point of early closure
  + The remaining balance at that point
  + The total interest loss incurred due to early repayment
* Distributed this interest loss over the remaining months to quantify the monthly impact of early closure.

**Formula:**

