Name : Ameya Gohad

PRN : 21070521007

Sec : A

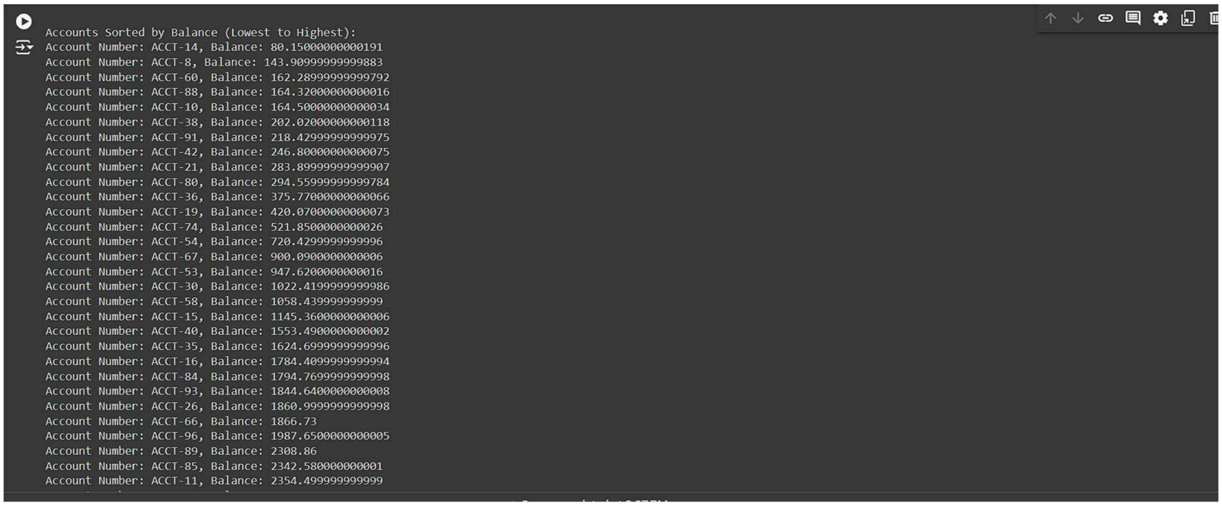
Sub : Gen AI

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.

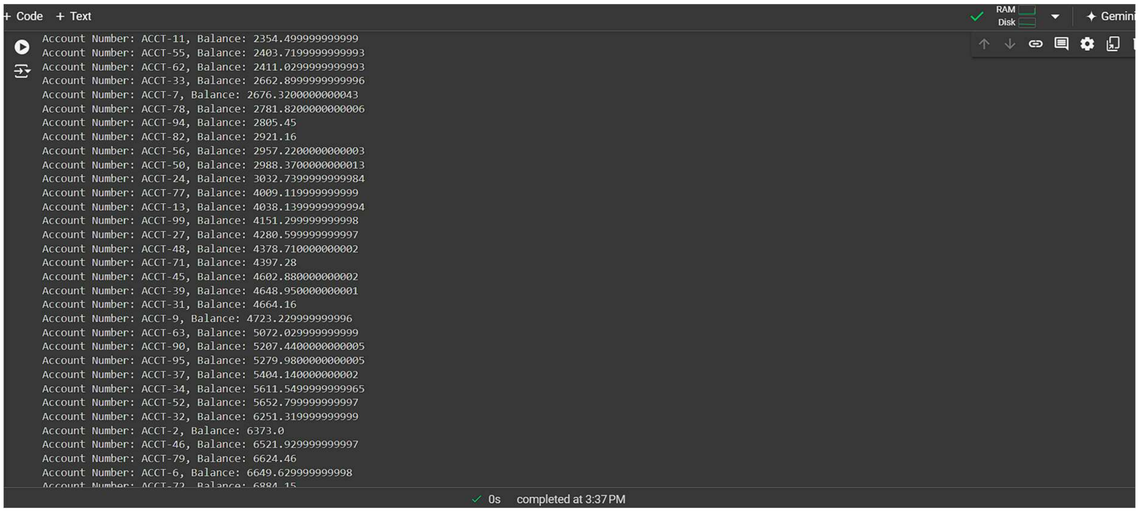
Sol :

```
import random

class BankAccount:
  def __init__(self, account_number, initial_balance):
    self.account_number = account_number
    self.balance = initial_balance
    self.transactions = []

  def deposit(self, amount):
    self.balance += amount
    self.transactions.append(('Deposit', amount))

  def withdraw(self, amount):
    if amount <= self.balance:
      self.balance -= amount
      self.transactions.append(('Withdraw', amount))
    else:
      self.transactions.append(('Withdraw', 0))
    def get_balance(self):
        return self.balance

    def print_account_summary(self):
        print(f"Account Number: {self.account_number}, Balance: {self.balance}")

def generate_accounts(num_accounts, months, num_transactions_per_month, seed_value):
    random.seed(seed_value)
    accounts = []

    for i in range(num_accounts):
        initial_balance = round(random.uniform(1000, 10000), 2)
        account = BankAccount(f"ACCT-{i+1}", initial_balance)

        for month in range(months):
            for _ in range(num_transactions_per_month):
                transaction_type = random.choice(['deposit', 'withdrawal'])
                amount = round(random.uniform(100, 1000), 2)

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

        accounts.append(account)

    return accounts

def print_sorted_accounts(accounts):
    sorted_accounts = sorted(accounts, key=lambda acc: acc.get_balance())
    print("\nAccounts Sorted by Balance (Lowest to Highest):")
    for account in sorted_accounts:
        account.print_account_summary()


num_accounts = 100
months = 6
num_transactions_per_month = 10
seed_value = 42


accounts = generate_accounts(num_accounts, months, num_transactions_per_month,
seed_value)


print_sorted_accounts(accounts)
```

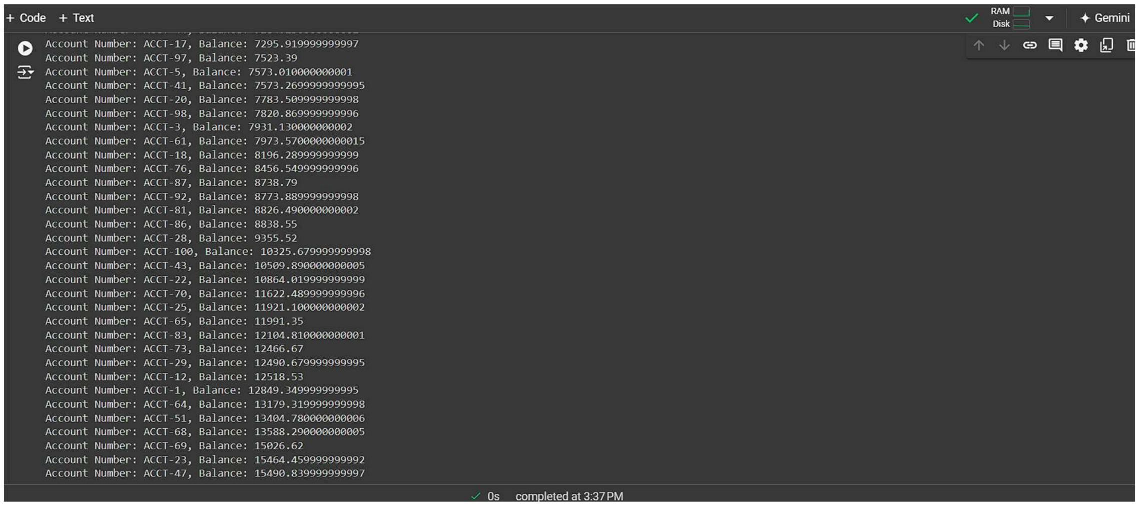
Output :



```
Accounts Sorted by Balance (Lowest to Highest):
Account Number: ACCT-14, Balance: 80.1500000000191
Account Number: ACCT-8, Balance: 143.90999999999883
Account Number: ACCT-60, Balance: 162.28999999999792
Account Number: ACCT-88, Balance: 164.32000000000016
Account Number: ACCT-10, Balance: 164.50000000000034
Account Number: ACCT-38, Balance: 202.02000000000118
Account Number: ACCT-91, Balance: 218.42999999999975
Account Number: ACCT-42, Balance: 246.80000000000075
Account Number: ACCT-21, Balance: 283.89999999999907
Account Number: ACCT-80, Balance: 294.55999999999784
Account Number: ACCT-36, Balance: 375.77000000000066
Account Number: ACCT-19, Balance: 420.07000000000073
Account Number: ACCT-74, Balance: 521.8500000000026
Account Number: ACCT-54, Balance: 720.4299999999996
Account Number: ACCT-67, Balance: 900.0900000000006
Account Number: ACCT-53, Balance: 947.6200000000016
Account Number: ACCT-30, Balance: 1022.4199999999986
Account Number: ACCT-58, Balance: 1058.439999999999
Account Number: ACCT-15, Balance: 1145.3600000000006
Account Number: ACCT-40, Balance: 1553.4900000000002
Account Number: ACCT-35, Balance: 1624.6999999999996
Account Number: ACCT-16, Balance: 1784.4099999999994
Account Number: ACCT-84, Balance: 1794.7699999999998
Account Number: ACCT-93, Balance: 1844.6400000000008
Account Number: ACCT-26, Balance: 1860.9999999999998
Account Number: ACCT-66, Balance: 1866.73
Account Number: ACCT-96, Balance: 1987.6500000000005
Account Number: ACCT-89, Balance: 2308.86
Account Number: ACCT-85, Balance: 2342.580000000001
Account Number: ACCT-11, Balance: 2354.499999999999
```



```
Account Number: ACCT-11, Balance: 2354.499999999999
Account Number: ACCT-55, Balance: 2403.7199999999993
Account Number: ACCT-62, Balance: 2411.0299999999993
Account Number: ACCT-33, Balance: 2662.8999999999996
Account Number: ACCT-7, Balance: 2676.3200000000043
Account Number: ACCT-78, Balance: 2781.8200000000006
Account Number: ACCT-94, Balance: 2805.45
Account Number: ACCT-82, Balance: 2921.16
Account Number: ACCT-56, Balance: 2957.2200000000003
Account Number: ACCT-50, Balance: 2988.3700000000013
Account Number: ACCT-24, Balance: 3032.7399999999984
Account Number: ACCT-77, Balance: 4009.119999999999
Account Number: ACCT-13, Balance: 4038.1399999999994
Account Number: ACCT-99, Balance: 4151.299999999998
Account Number: ACCT-27, Balance: 4280.599999999997
Account Number: ACCT-48, Balance: 4378.710000000002
Account Number: ACCT-71, Balance: 4397.28
Account Number: ACCT-45, Balance: 4602.880000000002
Account Number: ACCT-39, Balance: 4648.950000000001
Account Number: ACCT-31, Balance: 4664.16
Account Number: ACCT-9, Balance: 4723.229999999996
Account Number: ACCT-63, Balance: 5072.029999999999
Account Number: ACCT-90, Balance: 5207.4400000000005
Account Number: ACCT-95, Balance: 5279.9800000000005
Account Number: ACCT-37, Balance: 5404.140000000002
Account Number: ACCT-34, Balance: 5611.5499999999965
Account Number: ACCT-52, Balance: 5652.799999999997
Account Number: ACCT-32, Balance: 6251.319999999999
Account Number: ACCT-2, Balance: 6373.0
Account Number: ACCT-46, Balance: 6521.929999999997
Account Number: ACCT-79, Balance: 6624.46
Account Number: ACCT-6, Balance: 6649.629999999998
Account Number: ACCT-72, Balance: 6884.15
```



```
Account Number: ACCT-17, Balance: 7295.919999999997
Account Number: ACCT-97, Balance: 7523.39
Account Number: ACCT-5, Balance: 7573.010000000001
Account Number: ACCT-41, Balance: 7573.2699999999995
Account Number: ACCT-20, Balance: 7783.509999999998
Account Number: ACCT-98, Balance: 7820.869999999996
Account Number: ACCT-3, Balance: 7931.130000000002
Account Number: ACCT-61, Balance: 7973.5700000000015
Account Number: ACCT-18, Balance: 8196.289999999999
Account Number: ACCT-76, Balance: 8456.549999999996
Account Number: ACCT-87, Balance: 8738.79
Account Number: ACCT-92, Balance: 8773.889999999998
Account Number: ACCT-81, Balance: 8826.490000000002
Account Number: ACCT-86, Balance: 8838.55
Account Number: ACCT-28, Balance: 9355.52
Account Number: ACCT-100, Balance: 10325.679999999998
Account Number: ACCT-43, Balance: 10509.890000000005
Account Number: ACCT-22, Balance: 10864.019999999999
Account Number: ACCT-70, Balance: 11622.489999999996
Account Number: ACCT-25, Balance: 11921.100000000002
Account Number: ACCT-65, Balance: 11991.35
Account Number: ACCT-83, Balance: 12104.810000000001
Account Number: ACCT-73, Balance: 12466.67
Account Number: ACCT-29, Balance: 12490.679999999995
Account Number: ACCT-12, Balance: 12518.53
Account Number: ACCT-1, Balance: 12849.349999999995
Account Number: ACCT-64, Balance: 13179.319999999998
Account Number: ACCT-51, Balance: 13404.780000000006
Account Number: ACCT-68, Balance: 13588.290000000005
Account Number: ACCT-69, Balance: 15026.62
Account Number: ACCT-23, Balance: 15464.459999999992
Account Number: ACCT-47, Balance: 15490.839999999997
```

Explanation of Code :

BankAccount Class:

Each account has an account_number, an initial_balance, and a list of transactions.

Methods deposit and withdraw are used to update the balance and record transactions.

The get_balance method returns the current balance.

print_account_summary is used to print the account number and balance.

generate_accounts Function:

This function creates a number of accounts (num_accounts), each with a random initial balance.

It simulates monthly transactions by randomly choosing whether to deposit or withdraw an amount for each account.

A seed value is used to make the random numbers reproducible.

print_sorted_accounts Function:

This function sorts the accounts by their final balance (from lowest to highest) and prints a summary for each account.

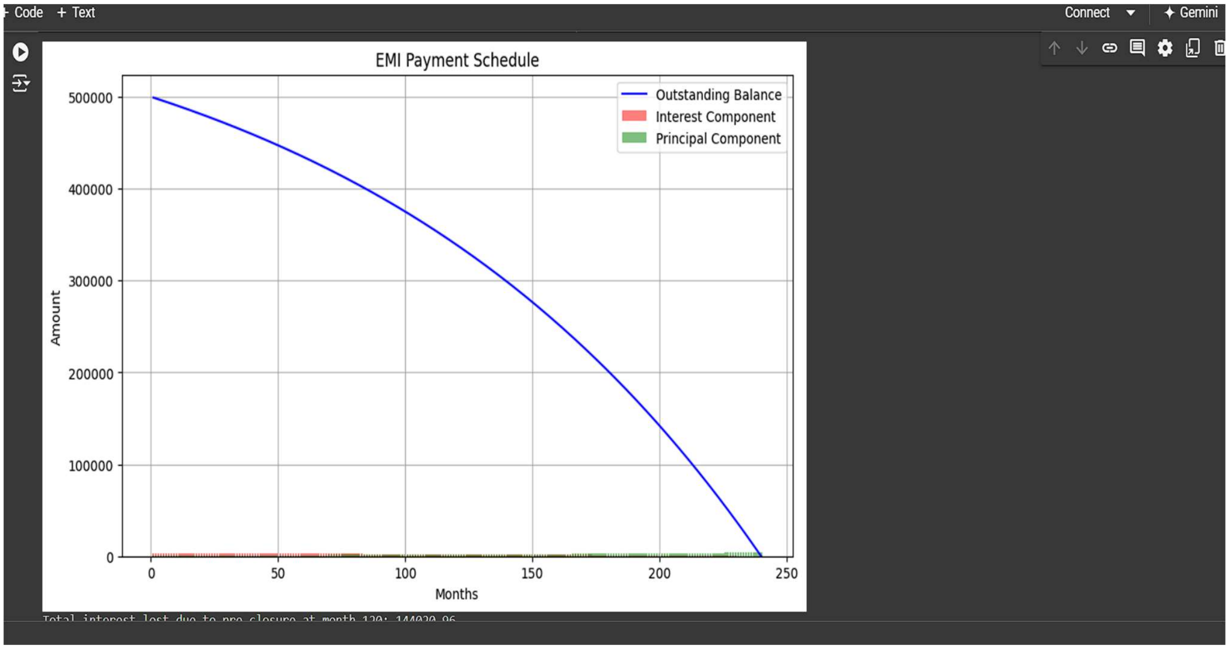
Q2 : Generate a model in Python to represent a Housing loan scheme and create a chart to display the Emi based on rate of interest and reducing balance for a given period. If a customer wishes to close the loan earlier, print the interest lost distributed over the remaining no. Of months. Assume suitable data and inputs as necessary.

Sol :

```
import numpy as np
import matplotlib.pyplot as plt


def calculate_emi(principal, annual_rate, tenure_years):
    monthly_rate = annual_rate / (12 * 100)
    tenure_months = tenure_years * 12

    emi = (principal * monthly_rate * (1 + monthly_rate)**tenure_months) / ((1 + monthly_rate)**tenure_months - 1)
    return emi, tenure_months


def generate_loan_schedule(principal, annual_rate, tenure_years, pre_close_month=None):
    emi, tenure_months = calculate_emi(principal, annual_rate, tenure_years)
    balance = principal
    interest_paid = 0
    emi_schedule = []

    for month in range(1, tenure_months + 1):
        monthly_interest = balance * (annual_rate / 12 / 100)
        principal_payment = emi - monthly_interest
        balance -= principal_payment
        interest_paid += monthly_interest

        emi_schedule.append((month, round(emi, 2), round(monthly_interest, 2), round(principal_payment, 2), round(balance, 2)))
        if pre_close_month is not None and month == pre_close_month:
            break

    return emi_schedule, interest_paid, tenure_months

def calculate_interest_lost(principal, annual_rate, tenure_years, pre_close_month):
    full_schedule, full_interest_paid, _ = generate_loan_schedule(principal, annual_rate, tenure_years)
    pre_close_schedule, pre_close_interest_paid, _ = generate_loan_schedule(principal, annual_rate, tenure_years, pre_close_month)

    interest_lost = full_interest_paid - pre_close_interest_paid
    return interest_lost, pre_close_schedule

def plot_emi_schedule(emi_schedule):
    months = [x[0] for x in emi_schedule]
    balances = [x[4] for x in emi_schedule]
    interests = [x[2] for x in emi_schedule]
    principals = [x[3] for x in emi_schedule]

    plt.figure(figsize=(10, 6))
    plt.plot(months, balances, label='Outstanding Balance', color='blue')
    plt.bar(months, interests, label='Interest Component', color='red', alpha=0.5)
    plt.bar(months, principals, label='Principal Component', color='green', alpha=0.5)

    plt.title('EMI Payment Schedule')
    plt.xlabel('Months')
    plt.ylabel('Amount')
    plt.legend()
    plt.grid(True)
    plt.show()

principal = 500000
```

```
annual_rate = 7.5

tenure_years = 20

pre_close_month = 120


emi_schedule, total_interest_paid, _ = generate_loan_schedule(principal, annual_rate, tenure_years)

plot_emi_schedule(emi_schedule)


interest_lost, pre_close_schedule = calculate_interest_lost(principal, annual_rate, tenure_years, pre_close_month)


print(f"Total interest lost due to pre-closure at month {pre_close_month}: {round(interest_lost, 2)}")
```

Output :



```
Total interest lost due to pre-closure at month 120: 144020.96
```

Explanation of Code :

EMI Calculation:The calculate_emi function computes the EMI for a given loan amount, interest rate, and tenure.

Loan Schedule:The generate_loan_schedule function computes the detailed EMI schedule, breaking down each EMI into interest and principal components and updating the reducing balance.

Pre-Closure:The calculate_interest_lost function calculates the total interest lost due to pre-closure. It compares the total interest that would have been paid over the full tenure with the interest paid up to the pre-closure month.

Visualization:The plot_emi_schedule function visualizes the EMI schedule, showing how the interest and principal components change over time, along with the reducing balance.