PRN: 21070521050

Name: Pari Nagpal

GenAI Assignment

Q1: 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)
```

Q2: This model calculates the EMI (Equated Monthly Installment) for a housing loan based on a reducing balance method. The user provides the loan principal, annual interest rate, and tenure (in months). The program also computes the interest loss if the loan is closed early.

```
import matplotlib.pyplot as plt
def calculate_emi(principal, annual_rate, tenure_months):
    monthly_rate = annual_rate / 12 / 100
    emi = principal * monthly_rate * (1 + monthly_rate) ** tenure_months / ((1 + monthly_rate) ** tenure_months - 1)
    return emi

def calculate_reducing_balance_emi(principal, annual_rate, tenure_months):
    monthly_rate = annual_rate / 12 / 100
    balance = principal
    emi = calculate_emi(principal, annual_rate, tenure_months)

    payments = []
    for month in range(tenure_months):
        interest = balance * monthly_rate
        principal_paid = emi - interest
        balance -= principal_paid
        payments.append((emi, balance if balance > 0 else 0, interest))
    return payments

def plot_emi_chart(payments, tenure_months):
    months = list(range(1, tenure_months + 1))
    balances = [payment[1] for payment in payments]

    plt.plot(months, balances, label="Remaining Balance")
    plt.xlabel("Month")
    plt.ylabel("Remaining Balance")
    plt.title("EMI and Reducing Balance over Time")
    plt.legend()
    plt.show()

principal = float(input("Enter loan principal amount: "))
annual_rate = float(input("Enter annual interest rate (in %): "))
tenure_months = int(input("Enter loan tenure in months: "))

payments = calculate_reducing_balance_emi(principal, annual_rate, tenure_months)
plot_emi_chart(payments, tenure_months)

def early_closure_interest_loss(payments, close_after_months):
    interest_paid = sum([payment[2] for payment in payments[:close_after_months]])
    total_interest_if_full = sum([payment[2] for payment in payments])
    interest_loss = total_interest_if_full - interest_paid
    return interest_loss
```

```
close_after_months = int(input(f"Enter the month after which you plan to close the loan early (1-{tenure_months}): "))

if close_after_months <= tenure_months:
    interest_loss = early_closure_interest_loss(payments, close_after_months)
    print(f"Interest lost if closed early after {close_after_months} months: {interest_loss:.2f}")
else:
    print(f"Invalid input. The month should be within the tenure of {tenure_months} months.")
```

Example Output:

Enter loan principal amount: 800000
Enter annual interest rate (in %): 9
Enter loan tenure in months: 48
Enter the month after which you plan to close the loan early (1-48): 24
Interest lost if closed early after 24 months: 42022.95