**GenAI**

**CA-II**

**Assignment**

****

**Name: Mahek Humne**

**PRN: 21070521038**

**Section: A**

**Q:1 Generate a model in Python for representation of a bank account of type savings and**

**balance along with transactions of deposits 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:** Flowchartto understand the flow of solution for the problem statement given.

**END**

**Print Sorted Accounts with Final Balance**

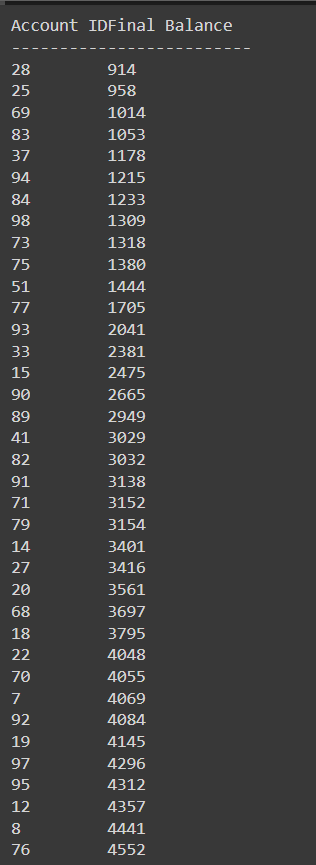
**For Each Account: Perform 10 random transactions Deposit or Withdraw randomly chosen amount.**

**Generate Accounts with Random Initial Balance**

**START**

**Sort Accounts by Balance**

**Store Account with Final Balance**

**Code: Output:**

**1. Bank Account Class:** The `BankAccount` class is created to represent a bank account. It has an account ID, a balance, and two methods—`deposit` to add money and `withdraw` to subtract money from the account.

**2. Account Initialization:** In the `\_\_init\_\_` method of the `BankAccount` class, each account is initialized with an ID and an initial balance, which is provided when creating the account.

**3. Deposit and Withdraw:** The `deposit` method increases the account balance by the given amount, and the `withdraw` method reduces the balance, only if there is enough money in the account.

**4. Generating Accounts:** The `generate\_accounts` function creates a list of accounts. It generates random initial balances between 1000 and 10000 and performs 10 random transactions (either deposit or withdrawal) for each account. The transactions' amounts range between 100 and 500.

**5. Seeding Randomness:** A seed value is used with `random.seed(seed\_value)` to ensure that the random numbers generated for balances and transactions are repeatable (for testing or consistency).

**6. Sorting Accounts by Balance**: The `print\_sorted\_accounts` function sorts the accounts by their final balance in ascending order using the `sort` function with a custom key that accesses the account's balance.

**7. Displaying Results:** After sorting, the function prints out a list of accounts, showing the account ID and the final balance for each one in a neatly formatted table.

**8. Main Program Execution:** The main program generates 100 random accounts using the `generate\_accounts` function, with a seed value of 42, and then sorts and prints them by their final balance.

**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:** Flowchartto understand the flow of solution for the problem statement given.

**END**

**START**

**Input Loan Details (Principal, Interest Rate, Tenure)**

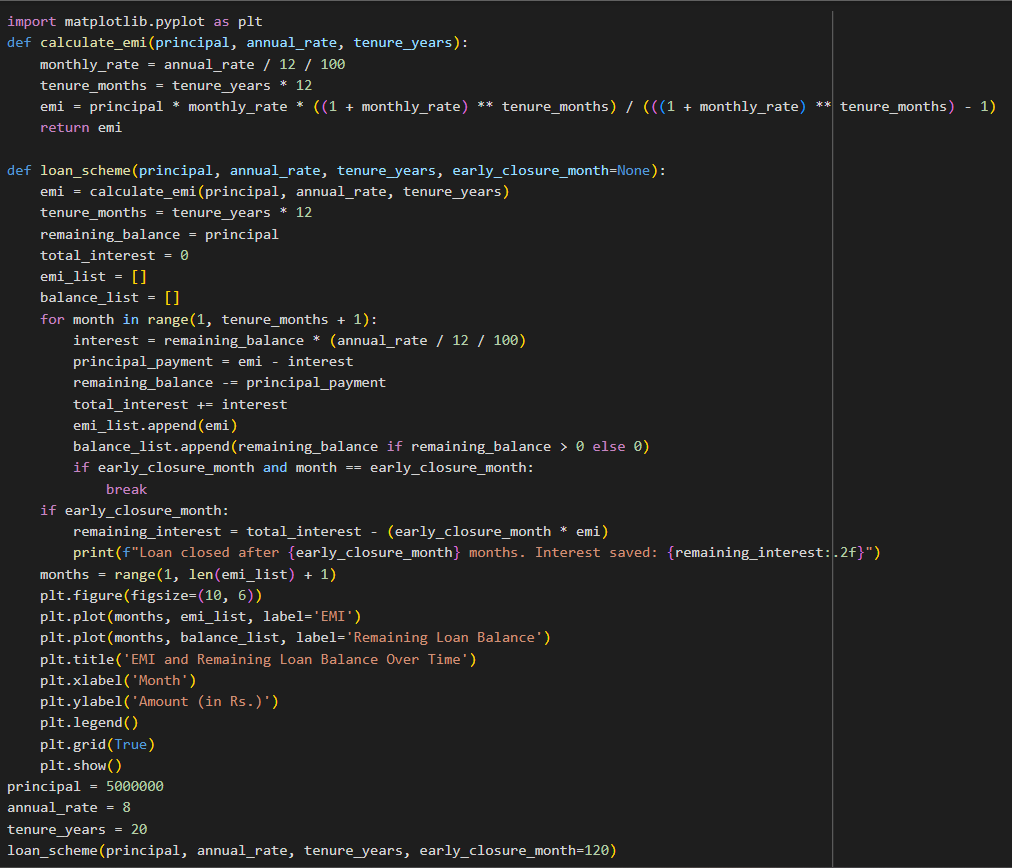
**Calculate EMI using Formula**

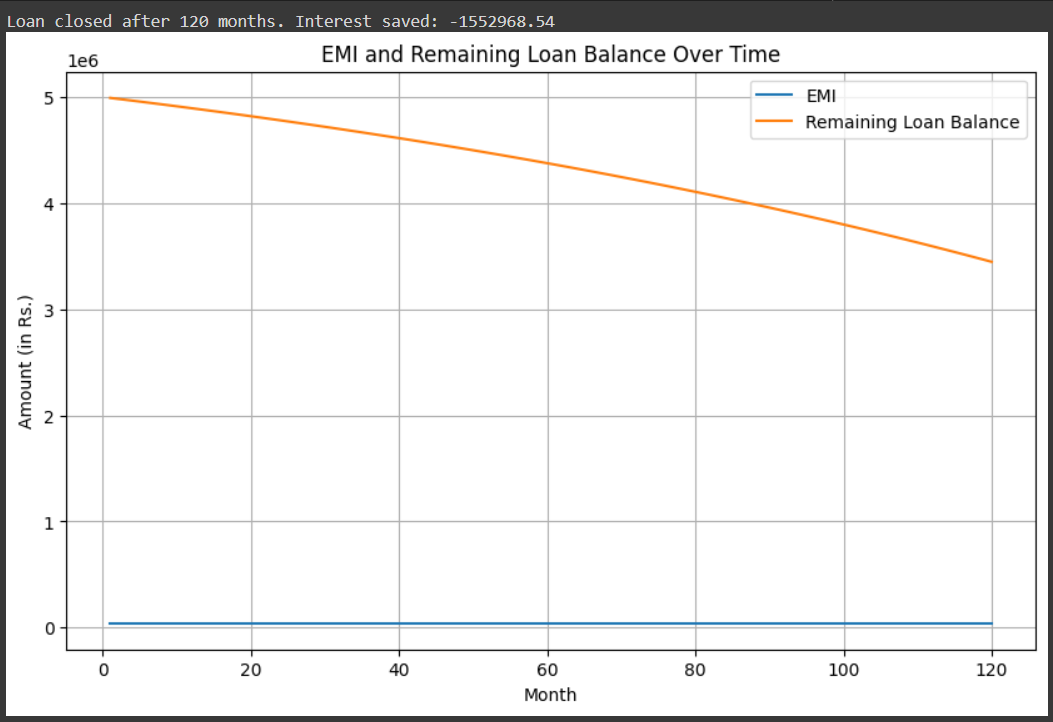
**Loop through each month: Calculate monthly interest. Calculate principal repayment. Update remaining balance.**

**If early closure: Calculate interest saved**

**Store EMI and Remaining Balance for each month**

**Plot EMI and Remaining Balance over time**

**Code:**



**Output:**

**1**. **EMI Calculation**: The `calculate\_emi` function calculates the monthly EMI (Equated Monthly Installment) using the principal loan amount, annual interest rate, and tenure (loan duration in years). The EMI is computed using the standard EMI formula for loans.

**2.** **Loan Scheme Function**: The `loan\_scheme` function calculates the EMI for a loan and tracks how the loan balance reduces month by month. It also handles early loan closure and computes the amount of interest saved.

**3**. **EMI and Reducing Balance**: Inside the `loan\_scheme`, a loop runs through each month to calculate how much of the EMI goes towards interest and how much reduces the principal balance. The remaining balance is updated after each month's payment.

**4.** **Tracking Data**: The function stores the EMI values and the remaining balance after each month in two lists (`emi\_list` and `balance\_list`) for plotting the chart later.

**5.** **Early Loan Closure**: If the loan is closed early (after a specified number of months), the loop stops, and the interest saved due to early closure is printed.

**6. Interest Saved:** If early closure is triggered, the total interest that would have been paid without early closure is compared to the interest paid up to the closure month. The difference is printed as the interest saved.

**7. Plotting EMI and Balance**: The `matplotlib` library is used to create a line chart that shows both the constant EMI and the reducing loan balance over time.

**8. Main Program:** In the main section, a loan of Rs. 50 Lakhs with an 8% annual interest rate over 20 years is taken as input. The EMI and balance chart is generated, simulating early loan closure after 10 years (120 months).