**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 from lowest to highest balance.

**Solution:** Bank Account Model with Random Transactions

Here I created a Python model to represent a savings bank account, simulate 100 accounts with random balances and transactions, and then sort them based on the final balance.

**Code:**

|  |
| --- |
| **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(('Deposit', amount))**    **def withdraw(self, amount):**  **if amount <= self.balance:**  **self.balance -= amount**  **self.transactions.append(('Withdraw', amount))**  **else:**  **print(f"Account {self.account\_id} has insufficient funds for withdrawal")**    **def \_\_str\_\_(self):**  **return f"Account ID: {self.account\_id}, Balance: {self.balance}, Transactions: {self.transactions}"**    **# Generate 100 accounts with random transactions**  **accounts = []**  **random.seed(100)**    **for i in range(100):**  **balance = random.randint(1000, 10000) # Initial random balance**  **account = BankAccount(account\_id=i+1, balance=balance)**    **# Simulate transactions for random months**  **num\_transactions = random.randint(5, 15) # Random number of transactions**  **for \_ in range(num\_transactions):**  **transaction\_type = random.choice(['Deposit', 'Withdraw'])**  **amount = random.randint(100, 2000)**  **if transaction\_type == 'Deposit':**  **account.deposit(amount)**  **else:**  **account.withdraw(amount)**    **accounts.append(account)**    **# Sort accounts by balance**  **sorted\_accounts = sorted(accounts, key=lambda x: x.balance)**    **# Display sorted accounts**  **for account in sorted\_accounts:**  **print(account)** |

Output:



This is the small sample of the output

**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.

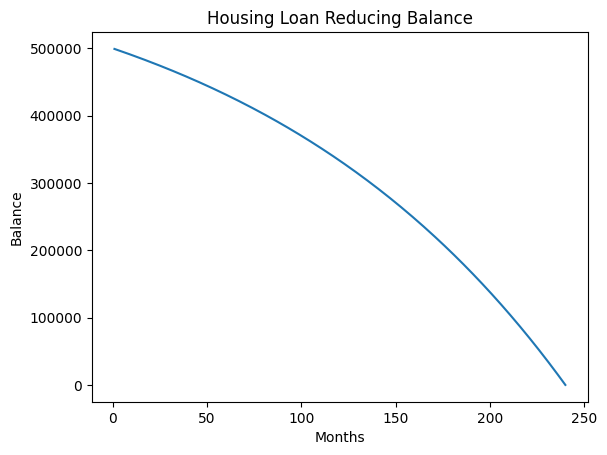
**Solution:** Housing Loan EMI and Early Closure Interest Calculation

Here I simulated a housing loan EMI calculation based on reducing balance and calculate the interest lost if the loan is closed early.

**Code:**

|  |
| --- |
| **import matplotlib.pyplot as plt**    **class HousingLoan:**  **def \_\_init\_\_(self, principal, rate\_of\_interest, tenure\_years):**  **self.principal = principal**  **self.rate\_of\_interest = rate\_of\_interest / 100 # Convert to decimal**  **self.tenure\_years = tenure\_years**  **self.tenure\_months = tenure\_years \* 12**  **self.emi = self.calculate\_emi()**    **def calculate\_emi(self):**  **monthly\_rate = self.rate\_of\_interest / 12**  **emi = (self.principal \* monthly\_rate \* ((1 + monthly\_rate) \*\* self.tenure\_months)) / (((1 + monthly\_rate) \*\* self.tenure\_months) - 1)**  **return emi**    **def early\_closure\_interest\_loss(self, closed\_months):**  **paid\_months = self.tenure\_months - closed\_months**  **outstanding\_balance = self.principal**  **total\_interest\_paid = 0**    **# Simulate EMI payments for paid months**  **for \_ in range(paid\_months):**  **interest = outstanding\_balance \* (self.rate\_of\_interest / 12)**  **principal\_repaid = self.emi - interest**  **outstanding\_balance -= principal\_repaid**  **total\_interest\_paid += interest**    **total\_interest\_saved = (self.emi \* self.tenure\_months) - total\_interest\_paid**  **return total\_interest\_saved**    **def plot\_emi(self):**  **balance = self.principal**  **months = []**  **balances = []**  **for month in range(1, self.tenure\_months + 1):**  **interest = balance \* (self.rate\_of\_interest / 12)**  **principal\_repaid = self.emi - interest**  **balance -= principal\_repaid**  **months.append(month)**  **balances.append(balance)**    **plt.plot(months, balances)**  **plt.title("Housing Loan Reducing Balance")**  **plt.xlabel("Months")**  **plt.ylabel("Balance")**  **plt.show()**    **# Example Usage**  **loan = HousingLoan(principal=500000, rate\_of\_interest=7, tenure\_years=20)**  **print(f"EMI: {loan.emi:.2f}")**  **loan.plot\_emi()**    **# Early closure after 10 years**  **months\_closed = 10 \* 12**  **interest\_saved = loan.early\_closure\_interest\_loss(months\_closed)**  **print(f"Interest saved if loan closed after {months\_closed} months: {interest\_saved:.2f}")** |

**Output:**

**zx**