**Symbiosis Institute of Technology, Nagpur**

**CA-II GenAI**

**Sub: GenAI Sem: VII**

**Name: MAYUR THAKRE PRN: 21070521041**

**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.**

**Ans:**

In this question need to make a model on housing loan scheme and calculate the EMI (Equated Monthly Installment) based on the rate of interest and a reducing balance method. The solution follows these steps:

1. Loan Inputs: The user inputs the principal amount, annual interest rate, and the loan term (in months).
2. EMI Calculation: Using the reducing balance formula, the EMI is computed. This method considers that interest is applied to the outstanding loan balance after each monthly payment, reducing the interest component as the balance decreases over time.
3. Simulating Loan Repayment: Each month, the program calculates:
4. Interest for that month based on the outstanding balance.
5. The principal repayment as the EMI minus the interest.
6. The outstanding balance is reduced by the principal payment.
7. Early Loan Closure: The program also allows for early closure of the loan. If the loan is closed early, the interest lost is calculated as the difference between the total interest paid with early closure and what would have been paid over the full term.
8. EMI Breakdown Chart: A chart is displayed showing the interest and principal components of the EMI over time, illustrating how the interest decreases as the outstanding loan balance reduces.

This solution dynamically calculates and visualizes the EMI and total interest payments, accounting for both regular and early loan closure scenarios.

CODE:

import math

import matplotlib.pyplot as plt

# Step 1: Define function to calculate EMI

def calculate\_emi(principal, rate\_of\_interest, loan\_term):

    monthly\_rate = rate\_of\_interest / (12 \* 100)  # Monthly interest rate

    emi = (principal \* monthly\_rate \* math.pow(1 + monthly\_rate, loan\_term)) / (math.pow(1 + monthly\_rate, loan\_term) - 1)

    return emi

# Step 2: Simulate loan repayment and calculate EMI and interest for each month

def simulate\_loan\_repayment(principal, rate\_of\_interest, loan\_term):

    emi = calculate\_emi(principal, rate\_of\_interest, loan\_term)

    remaining\_principal = principal

    interest\_paid = []

    principal\_paid = []

    for month in range(1, loan\_term + 1):

        monthly\_interest = remaining\_principal \* rate\_of\_interest / (12 \* 100)

        monthly\_principal = emi - monthly\_interest

        remaining\_principal -= monthly\_principal

        # Track interest and principal components

        interest\_paid.append(monthly\_interest)

        principal\_paid.append(monthly\_principal)

        if remaining\_principal <= 0:

            break

    return emi, interest\_paid, principal\_paid

# Step 3: Calculate interest lost if loan is closed early

def calculate\_early\_closure\_loss(interest\_paid, early\_closure\_month):

    total\_interest\_paid\_early = sum(interest\_paid[:early\_closure\_month])

    total\_interest\_paid\_full = sum(interest\_paid)

    interest\_lost = total\_interest\_paid\_full - total\_interest\_paid\_early

    return interest\_lost

# Step 4: Plot EMI chart showing interest and principal components

def plot\_emi\_chart(interest\_paid, principal\_paid):

    months = list(range(1, len(interest\_paid) + 1))

    plt.plot(months, interest\_paid, label='Interest Component')

    plt.plot(months, principal\_paid, label='Principal Component')

    plt.xlabel('Month')

    plt.ylabel('Amount Paid')

    plt.title('EMI Breakdown: Interest vs Principal')

    plt.legend()

    plt.grid(True)

    plt.show()

# Step 5: Main function to handle loan details and early closure

def main():

    # Input loan details

    principal = float(input("Enter the loan amount: "))

    rate\_of\_interest = float(input("Enter the annual interest rate (in %): "))

    loan\_term = int(input("Enter the loan term (in months): "))

    # Calculate EMI and simulate loan repayment

    emi, interest\_paid, principal\_paid = simulate\_loan\_repayment(principal, rate\_of\_interest, loan\_term)

    print(f"Monthly EMI: {emi:.2f}")

    # Plot the EMI breakdown chart

    plot\_emi\_chart(interest\_paid, principal\_paid)

    # Input for early loan closure

    early\_closure\_month = int(input("Enter the month when loan is closed early (0 for full term): "))

    if early\_closure\_month > 0:

        # Calculate interest loss for early closure

        interest\_lost = calculate\_early\_closure\_loss(interest\_paid, early\_closure\_month)

        print(f"Interest lost due to early closure: {interest\_lost:.2f}")

    else:

        print("Loan was repaid over the full term.")

# Run the main function

if \_\_name\_\_ == "\_\_main\_\_": # Corrected the typo in the if statement

    main() # Added indentation to ensure the main function is called within the if block

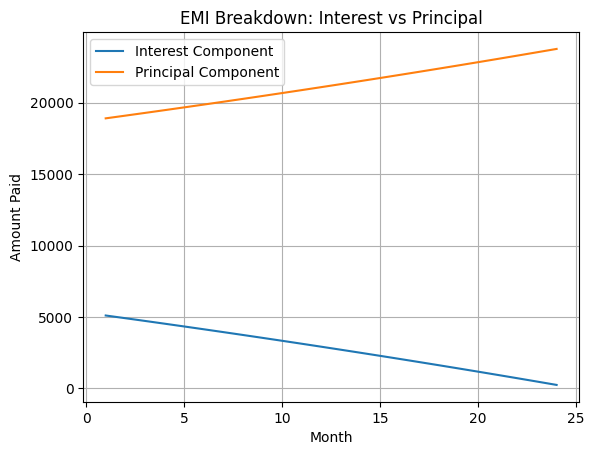
OUTPUT:

Enter the loan amount: 510000

Enter the annual interest rate (in %): 12

Enter the loan term (in months): 24

Monthly EMI: 24007.47



**Q:4 Generate a model to represent interest calculations of a Bank account where the process of**

**calculating interest for 6 months is a. Find minimum balance for each month b. Make a total of**

**all minimum balances c. Calculate interest based on interest rate d. Divide interest by 12 to**

**find one-month interest e. Multiply interest by 6 to show interest in the account. Generate a**

**model to represent transactions and interest calculations for 6 months.**

**Solution🡪**

import pandas as pd

import numpy as np

class BankAccount:

def \_\_init\_\_(self, account\_number, interest\_rate):

self.account\_number = account\_number

self.interest\_rate = interest\_rate # annual interest rate (e.g., 5% as 0.05)

self.transactions = {month: [] for month in range(1, 7)} # dictionary to store transactions for each month

def add\_transaction(self, month, amount):

if month in self.transactions:

self.transactions[month].append(amount)

else:

print(f"Invalid month {month}. Only 1-6 are valid.")

def find\_minimum\_balance(self):

min\_balances = {}

balance = 0

for month in range(1, 7):

monthly\_transactions = self.transactions[month]

monthly\_balances = [balance + sum(monthly\_transactions[:i + 1]) for i in range(len(monthly\_transactions))]

if monthly\_balances:

min\_balances[month] = min(monthly\_balances)

balance = monthly\_balances[-1] # Carry forward the last balance to the next month

else:

min\_balances[month] = balance # No transactions, so balance carries over

return min\_balances

def calculate\_interest(self):

min\_balances = self.find\_minimum\_balance()

total\_min\_balance = sum(min\_balances.values())

yearly\_interest = total\_min\_balance \* self.interest\_rate

monthly\_interest = yearly\_interest / 12

# Step d & e: Calculate interest for 6 months

six\_month\_interest = monthly\_interest \* 6

return six\_month\_interest, min\_balances

def generate\_transaction\_summary(self):

max\_len = max(len(transactions) for transactions in self.transactions.values())

padded\_transactions = {

month: transactions + [np.nan] \* (max\_len - len(transactions))

for month, transactions in self.transactions.items()

}

summary = pd.DataFrame(padded\_transactions)

summary.index.name = 'Transaction\_Index'

return summary.T # Transpose to make months the index

account = BankAccount(account\_number="123456789", interest\_rate=0.05) # 5% annual interest

account.add\_transaction(1, 1000)

account.add\_transaction(1, -200)

account.add\_transaction(1, 300)

account.add\_transaction(2, 1500)

account.add\_transaction(2, -500)

account.add\_transaction(3, 100)

account.add\_transaction(3, -50)

account.add\_transaction(4, 1200)

account.add\_transaction(4, -300)

account.add\_transaction(5, 400)

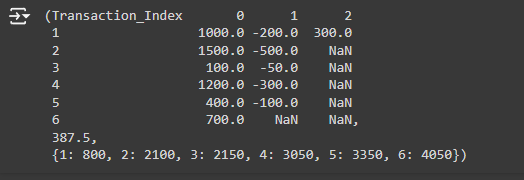
account.add\_transaction(5, -100)

account.add\_transaction(6, 700)

transaction\_summary = account.generate\_transaction\_summary()

six\_month\_interest, min\_balances = account.calculate\_interest()

transaction\_summary, six\_month\_interest, min\_balances



**Explaination:**

BankAccount Class:

The class contains methods for adding transactions (add\_transaction), finding the minimum balance for each month (find\_minimum\_balance), and calculating the interest for 6 months (calculate\_interest).

Transactions:

Transactions are stored month by month (1-6).

The minimum balance for each month is calculated based on the transactions.

Interest Calculation:

Once the minimum balance for each month is determined, the total minimum balance is calculated.

The yearly interest is computed by multiplying the total balance by the interest rate, then divided by 12 to get monthly interest.

The final interest for 6 months is obtained by multiplying the monthly interest by 6.

Result:

The output will show the transaction summary, the calculated interest for 6 months, and the minimum balances for each month.