Name: Somesh Nagar **PRN:** 21070521080

Department: CSE **Division:** B

Symbiosis Institute of Technology
Department of Applied Science

॥वसुधैव कुटुम्बकम्॥

Generative AI
CA-2

Computer Science and Engineering

Batch 2021-2025

Semester - VII

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.

The Python model simulates the housing loan process. The model consists of several key components:

1. Principal (Loan Amount): The total amount borrowed by the borrower.
2. Annual Interest Rate: The interest rate at which the loan is offered.
3. Tenure: The period the loan is repaid, typically measured in years.
4. EMI Calculation: The monthly instalment that includes both the repayment of the loan principal and the interest. The EMI is computed using the formula for reducing balance loans:

$$EMI = \frac{P \times r \times (1+r)^n}{(1+r)^n - 1}$$

`

Code:

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

class HousingLoanScheme:
    def __init__(self, principal, annual_rate, tenure_years):
        self.principal = principal
        self.annual_rate = annual_rate / 100
        self.tenure_months = tenure_years * 12
        self.monthly_rate = self.annual_rate / 12

    def calculate_emi(self):
        """Calculate EMI using the formula for reducing balance method"""
        P = self.principal
        r = self.monthly_rate
        n = self.tenure_months
        emi = (P * r * (1 + r)**n) / ((1 + r)**n - 1)
        return emi
```

__init__(self, principal, annual_rate, tenure_years)
This is the constructor of the HousingLoanScheme class. It initialises the loan parameters:

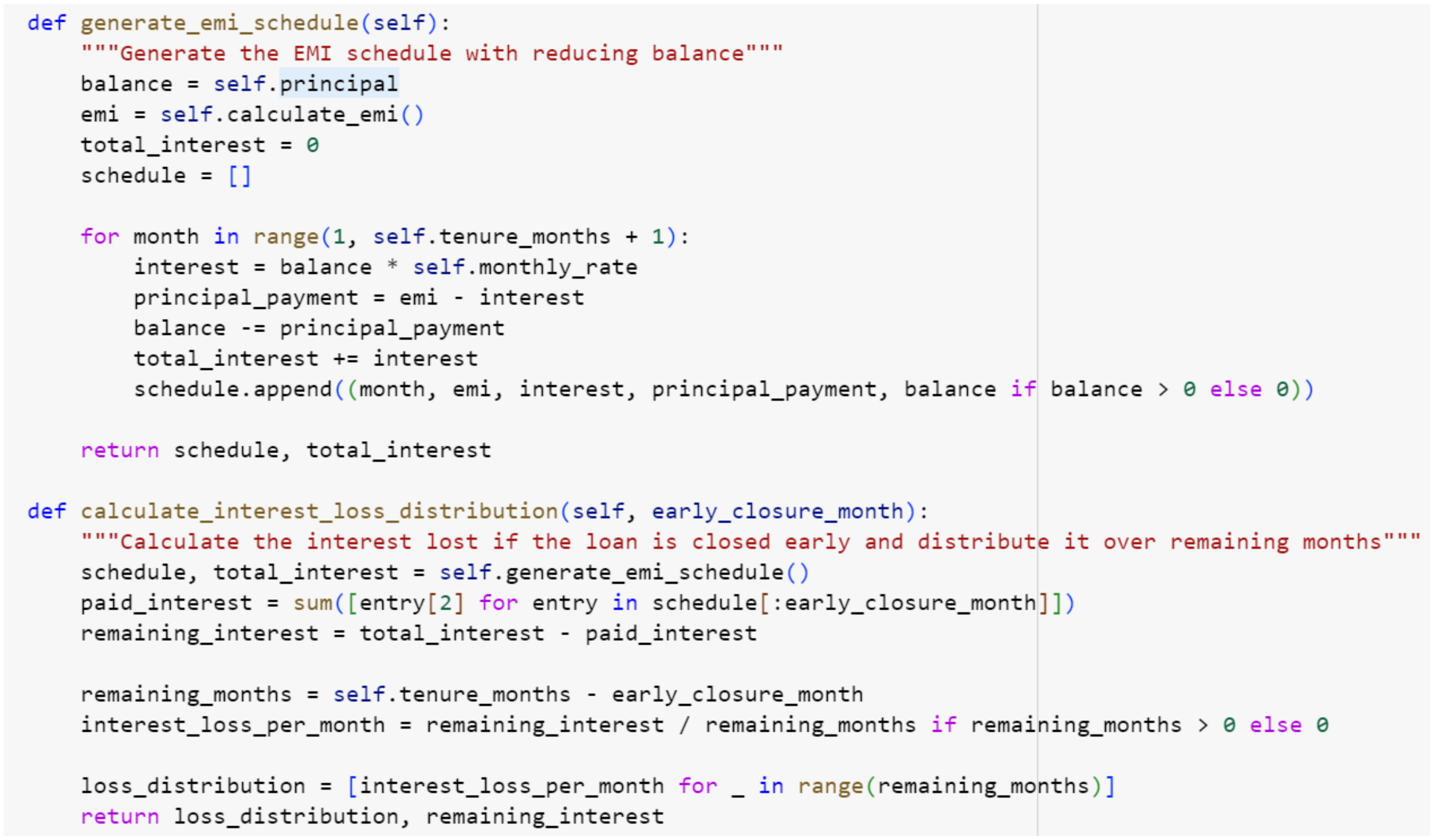
- principal: The loan amount (in currency units).
- annual_rate: The annual interest rate as a percentage (converted to a decimal value).
- tenure_months: The total loan tenure in months (years * 12).
- monthly_rate: The monthly interest rate is calculated by dividing the annual rate by 12.

calculate_emi(self)
This method calculates the Equated Monthly Installment (EMI) using the reducing balance method.

- P is the loan principal (initial amount).
- r is the monthly interest rate.
- n is the total number of months for repayment.

The EMI formula is:

$$EMI = \frac{P \times r \times (1+r)^n}{(1+r)^n - 1}$$



```python
def generate_emi_schedule(self):
    """Generate the EMI schedule with reducing balance"""
    balance = self.principal
    emi = self.calculate_emi()
    total_interest = 0
    schedule = []

    for month in range(1, self.tenure_months + 1):
        interest = balance * self.monthly_rate
        principal_payment = emi - interest
        balance -= principal_payment
        total_interest += interest
        schedule.append((month, emi, interest, principal_payment, balance if balance > 0 else 0))

    return schedule, total_interest

def calculate_interest_loss_distribution(self, early_closure_month):
    """Calculate the interest lost if the loan is closed early and distribute it over remaining months"""
    schedule, total_interest = self.generate_emi_schedule()
    paid_interest = sum([entry[2] for entry in schedule[:early_closure_month]])
    remaining_interest = total_interest - paid_interest

    remaining_months = self.tenure_months - early_closure_month
    interest_loss_per_month = remaining_interest / remaining_months if remaining_months > 0 else 0

    loss_distribution = [interest_loss_per_month for _ in range(remaining_months)]
    return loss_distribution, remaining_interest
```

generate_emi_schedule(self)
This method generates a schedule of EMI payments over the loan tenure, splitting each EMI into the portion paid toward **interest** and the portion paid toward the **principal**.

- It uses a monthly loop in the loan tenure (tenure_months).
- In each iteration, the following is calculated:
    - interest: The interest for that month (balance * montly_rate).
    - principal_payment: The part of the EMI that reduces the principal.
    - balance: The remaining loan balance after the principal is paid.

The function returns:

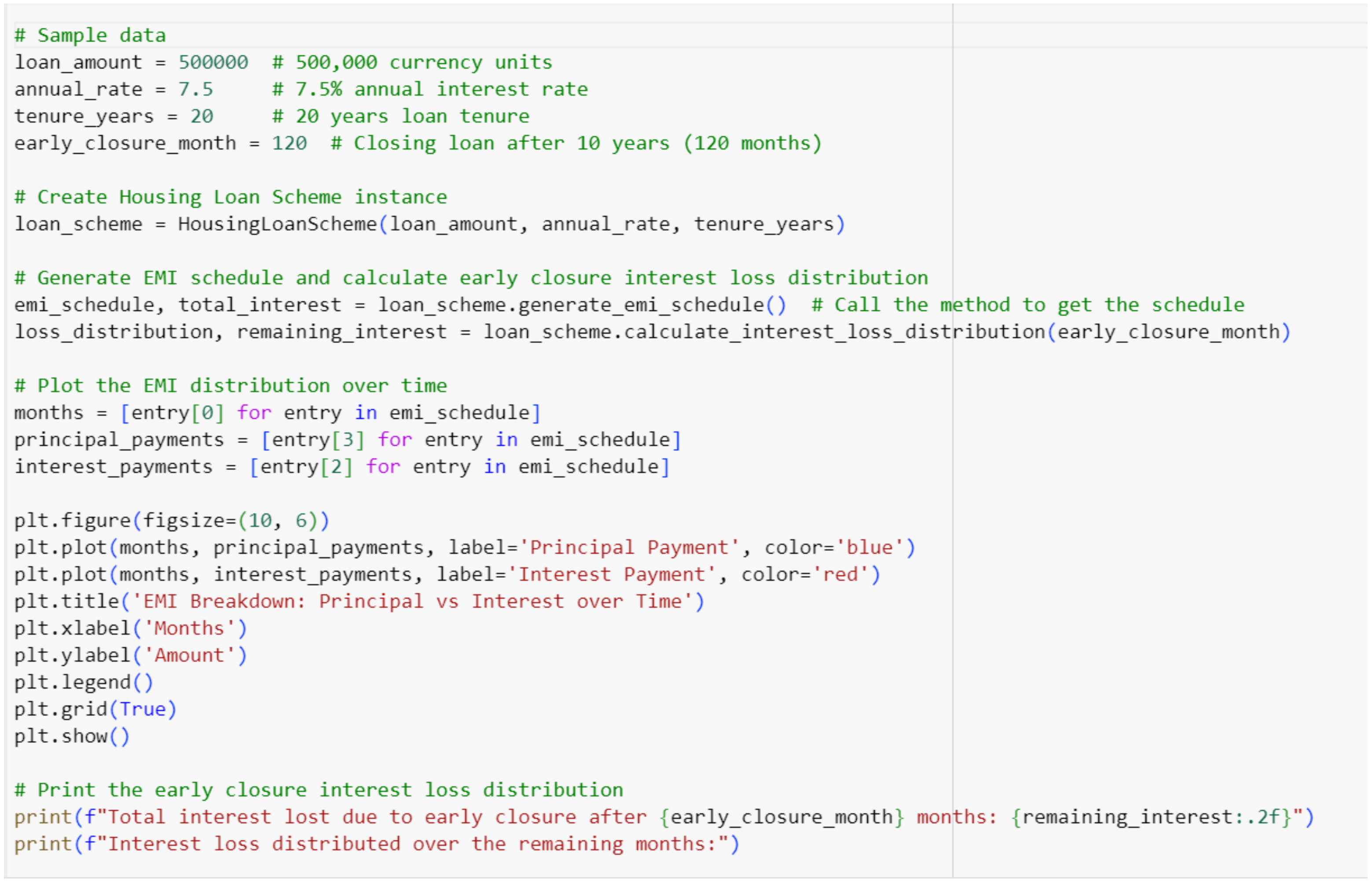
- A **schedule** list containing tuples for each month (month number, EMI, interest, principal payment, remaining balance).
- The **total interest** paid over the life of the loan.

calculate_interest_loss_distribution(self, early_closure_month)
This method calculates the **interest lost** due to early loan closure and distributes it over the remaining loan tenure.
Steps:

- It first generates the EMI schedule using generate_emi_schedule().
- Then, it sums the interest paid up until the specified **early closure month**.
- The remaining interest (that would have been paid if the loan were continued) is calculated as:
  remaining_interest=total_interest−paid_interest\text{remaining_interest} = \text{total_interest} - \text{paid_interest}remaining_interest=total_interest−paid_interest
- The remaining interest is distributed evenly over the months that would have been left if the loan were not closed early.

This method returns:

- The **loss distribution**: a list of how much interest is lost for each month.
- The **remaining interest** that was avoided by closing the loan early.



```python
# Sample data
loan_amount = 500000  # 500,000 currency units
annual_rate = 7.5     # 7.5% annual interest rate
tenure_years = 20     # 20 years loan tenure
early_closure_month = 120  # Closing loan after 10 years (120 months)

# Create Housing Loan Scheme instance
loan_scheme = HousingLoanScheme(loan_amount, annual_rate, tenure_years)

# Generate EMI schedule and calculate early closure interest loss distribution
emi_schedule, total_interest = loan_scheme.generate_emi_schedule()  # Call the method to get the schedule
loss_distribution, remaining_interest = loan_scheme.calculate_interest_loss_distribution(early_closure_month)

# Plot the EMI distribution over time
months = [entry[0] for entry in emi_schedule]
principal_payments = [entry[3] for entry in emi_schedule]
interest_payments = [entry[2] for entry in emi_schedule]

plt.figure(figsize=(10, 6))
plt.plot(months, principal_payments, label='Principal Payment', color='blue')
plt.plot(months, interest_payments, label='Interest Payment', color='red')
plt.title('EMI Breakdown: Principal vs Interest over Time')
plt.xlabel('Months')
plt.ylabel('Amount')
plt.legend()
plt.grid(True)
plt.show()

# Print the early closure interest loss distribution
print(f"Total interest lost due to early closure after {early_closure_month} months: {remaining_interest:.2f}")
print(f"Interest loss distributed over the remaining months:")
```

Plotting the EMI Breakdown

The code generates a plot showing how the EMI is divided into **interest** and **principal payments** over time:

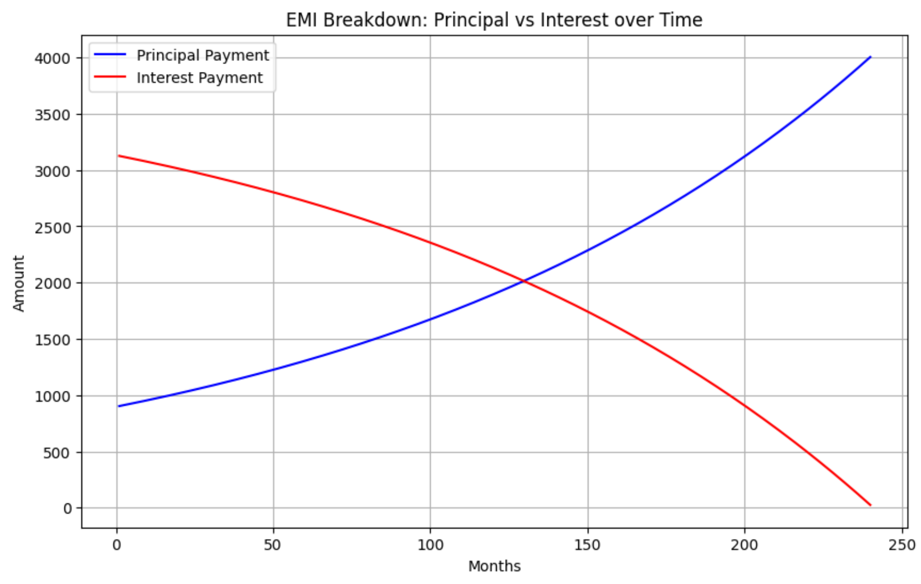
- The x-axis represents the number of months.
- The y-axis represents the payment amount.
- Two lines are plotted:
  - The **blue** line for the principal payments.
  - The **red** line for the interest payments.

This shows how, over time, the interest portion decreases while the principal payment increases (because of the reducing balance method).

Print the Early Closure Interest Loss Distribution

The code prints how much interest is lost each month after the loan is closed early. It shows the loss of interest every month after the early closure point, making it clear how much interest is saved by closing the loan early.

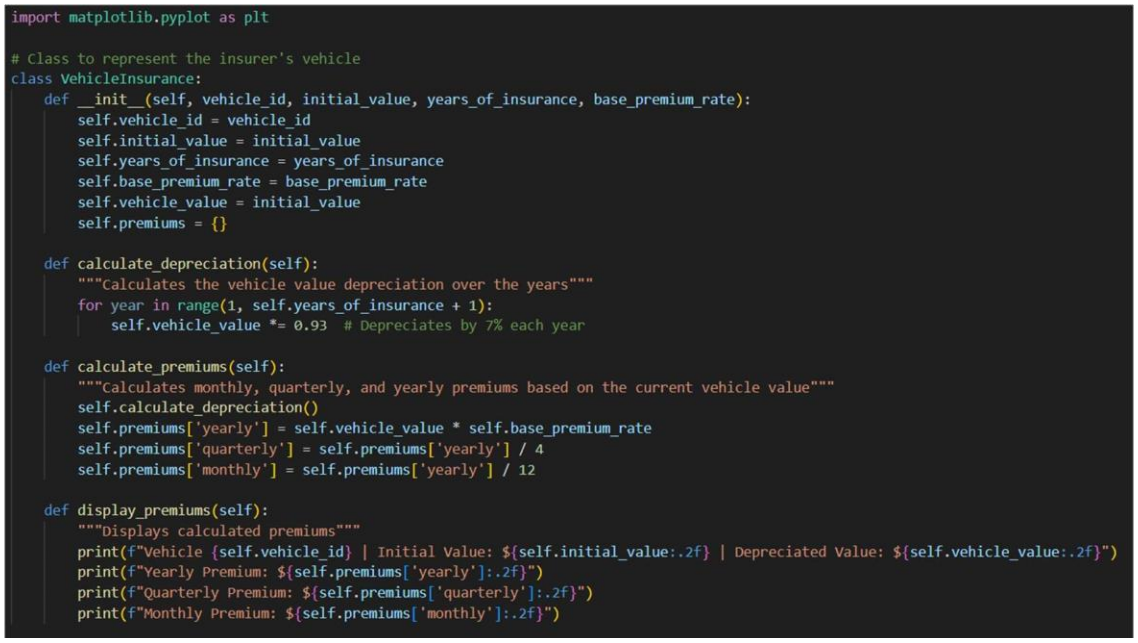
Output (Generated Data):



```
Total interest lost due to early closure after 120 months: 144020.96
Interest loss distributed over the remaining months:
Month 121: Lost interest = 1200.17
Month 122: Lost interest = 1200.17
Month 123: Lost interest = 1200.17
Month 124: Lost interest = 1200.17
Month 125: Lost interest = 1200.17
Month 126: Lost interest = 1200.17
Month 127: Lost interest = 1200.17
Month 128: Lost interest = 1200.17
Month 129: Lost interest = 1200.17
Month 130: Lost interest = 1200.17
Month 131: Lost interest = 1200.17
Month 132: Lost interest = 1200.17
Month 133: Lost interest = 1200.17
Month 134: Lost interest = 1200.17
Month 135: Lost interest = 1200.17
Month 136: Lost interest = 1200.17
```

Q.3 Generate a model for an Insurance company to hold information on the insurer's vehicle, and create a chart of monthly, yearly, and quarterly premiums based on no. ofyears of insurance where in each year, the value of the vehicle depreciates by 7%.

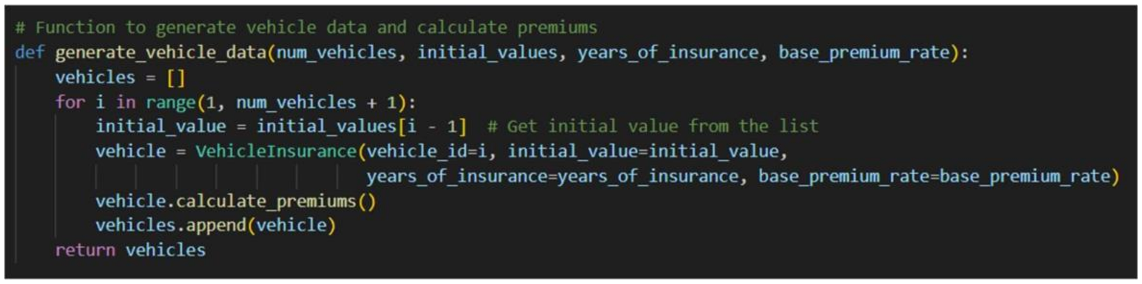
Code:



```python
import matplotlib.pyplot as plt

# Class to represent the insurer's vehicle
class VehicleInsurance:
    def __init__(self, vehicle_id, initial_value, years_of_insurance, base_premium_rate):
        self.vehicle_id = vehicle_id
        self.initial_value = initial_value
        self.years_of_insurance = years_of_insurance
        self.base_premium_rate = base_premium_rate
        self.vehicle_value = initial_value
        self.premiums = {}

    def calculate_depreciation(self):
        """Calculates the vehicle value depreciation over the years"""
        for year in range(1, self.years_of_insurance + 1):
            self.vehicle_value *= 0.93  # Depreciates by 7% each year

    def calculate_premiums(self):
        """Calculates monthly, quarterly, and yearly premiums based on the current vehicle value"""
        self.calculate_depreciation()
        self.premiums['yearly'] = self.vehicle_value * self.base_premium_rate
        self.premiums['quarterly'] = self.premiums['yearly'] / 4
        self.premiums['monthly'] = self.premiums['yearly'] / 12

    def display_premiums(self):
        """Displays calculated premiums"""
        print(f"Vehicle {self.vehicle_id} | Initial Value: ${self.initial_value:.2f} | Depreciated Value: ${self.vehicle_value:.2f}")
        print(f"Yearly Premium: ${self.premiums['yearly']:.2f}")
        print(f"Quarterly Premium: ${self.premiums['quarterly']:.2f}")
        print(f"Monthly Premium: ${self.premiums['monthly']:.2f}")
```

Class VehicleInsurance**:**

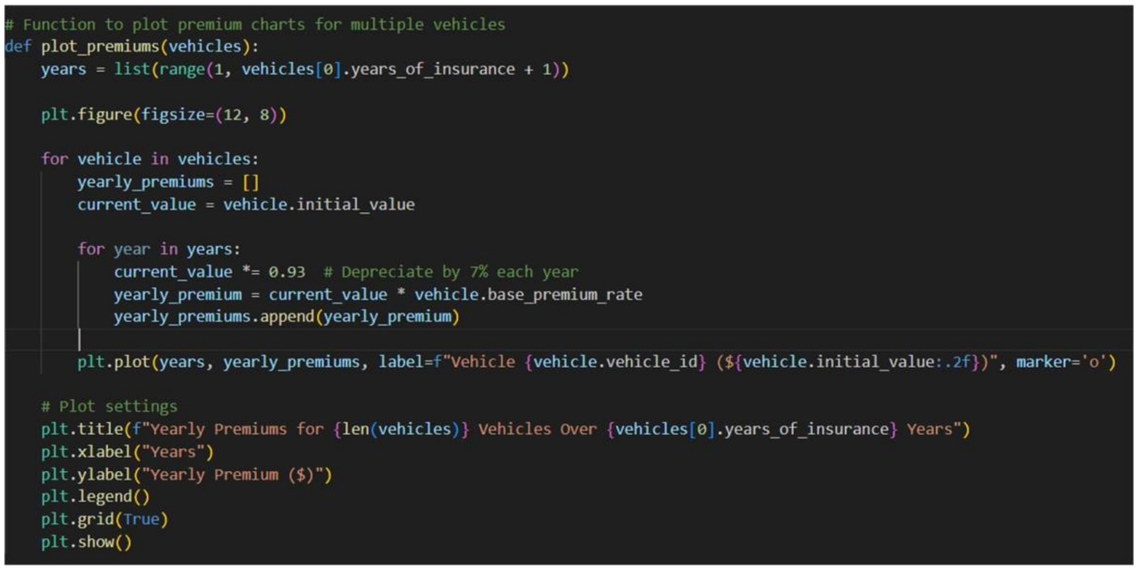
- This class models a vehicle with attributes for the vehicle ID, initial value, number of years of insurance, base premium rate, and a dictionary for storing premium values (monthly, quarterly,yearly).
- It has methods to calculate the depreciation (7% per year), calculate the premiums based onthe depreciated value, display the premiums for the vehicle.



```python
# Function to generate vehicle data and calculate premiums
def generate_vehicle_data(num_vehicles, initial_values, years_of_insurance, base_premium_rate):
    vehicles = []
    for i in range(1, num_vehicles + 1):
        initial_value = initial_values[i - 1]  # Get initial value from the list
        vehicle = VehicleInsurance(vehicle_id=i, initial_value=initial_value,
                                   years_of_insurance=years_of_insurance, base_premium_rate=base_premium_rate)
        vehicle.calculate_premiums()
        vehicles.append(vehicle)
    return vehicles
```

Function to generate the vehicle data generate_vehicle_data**:**

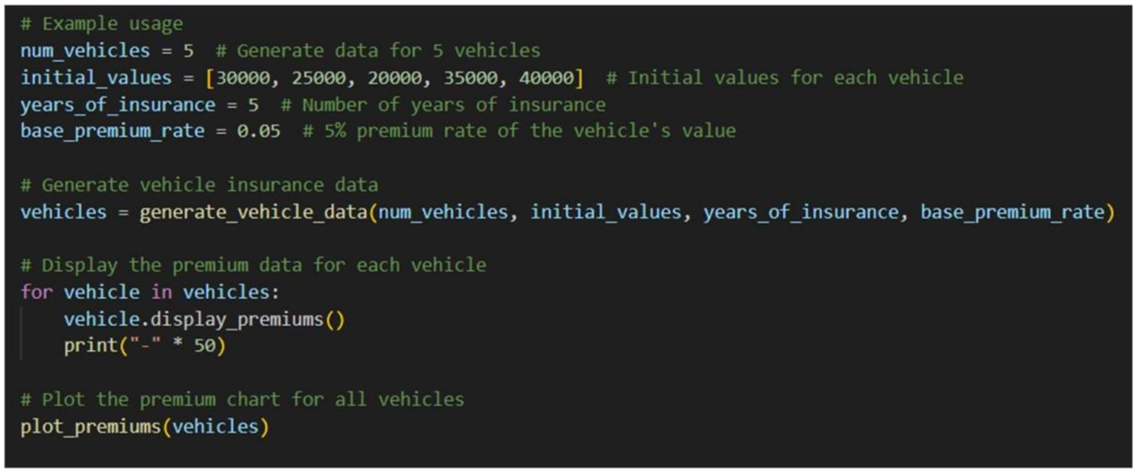
- Generates data for multiple vehicles, but for simplicity, it's generating one vehicle here.



```python
# Function to plot premium charts for multiple vehicles
def plot_premiums(vehicles):
    years = list(range(1, vehicles[0].years_of_insurance + 1))

    plt.figure(figsize=(12, 8))

    for vehicle in vehicles:
        yearly_premiums = []
        current_value = vehicle.initial_value

        for year in years:
            current_value *= 0.93  # Depreciate by 7% each year
            yearly_premium = current_value * vehicle.base_premium_rate
            yearly_premiums.append(yearly_premium)

        plt.plot(years, yearly_premiums, label=f"Vehicle {vehicle.vehicle_id} (${vehicle.initial_value:.2f})", marker='o')

    # Plot settings
    plt.title(f"Yearly Premiums for {len(vehicles)} Vehicles Over {vehicles[0].years_of_insurance} Years")
    plt.xlabel("Years")
    plt.ylabel("Yearly Premium ($)")
    plt.legend()
    plt.grid(True)
    plt.show()
```

Function to plot the graphs plot_premiums**:**

- Plots a line chart showing how the premiums (monthly, quarterly, and yearly) change over theyears as the vehicle value depreciates.



```python
# Example usage
num_vehicles = 5  # Generate data for 5 vehicles
initial_values = [30000, 25000, 20000, 35000, 40000]  # Initial values for each vehicle
years_of_insurance = 5  # Number of years of insurance
base_premium_rate = 0.05  # 5% premium rate of the vehicle's value

# Generate vehicle insurance data
vehicles = generate_vehicle_data(num_vehicles, initial_values, years_of_insurance, base_premium_rate)

# Display the premium data for each vehicle
for vehicle in vehicles:
    vehicle.display_premiums()
    print("-" * 50)

# Plot the premium chart for all vehicles
plot_premiums(vehicles)
```

Depreciation and Premium Calculation:

- The vehicle's value depreciates by 7% per year.
- Premiums are calculated as a percentage of the vehicle's current value after depreciation (5%in this example).
- Monthly and quarterly premiums are derived from the yearly premium.

Output (Generated Data):

The console will display the initial vehicle value, the depreciated value after 5 years, and the calculatedpremiums (monthly, quarterly, yearly).

```
Vehicle 1 | Initial Value: $30000.00 | Depreciated Value: $20870.65

Yearly Premium: $1043.53

Quarterly Premium: $260.88

Monthly Premium: $86.96

--------------------------------------------------

Vehicle 2 | Initial Value: $25000.00 | Depreciated Value: $17392.21

Yearly Premium: $869.61

Quarterly Premium: $217.40

Monthly Premium: $72.47

--------------------------------------------------

Vehicle 3 | Initial Value: $20000.00 | Depreciated Value: $13913.77

Yearly Premium: $695.69

Quarterly Premium: $173.92

Monthly Premium: $57.97

--------------------------------------------------

Vehicle 4 | Initial Value: $35000.00 | Depreciated Value: $24349.09

Yearly Premium: $1217.45

Quarterly Premium: $304.36

Monthly Premium: $101.45

--------------------------------------------------
```

Vehicle 5 | Initial Value: $40000.00 | Depreciated Value: $27827.53

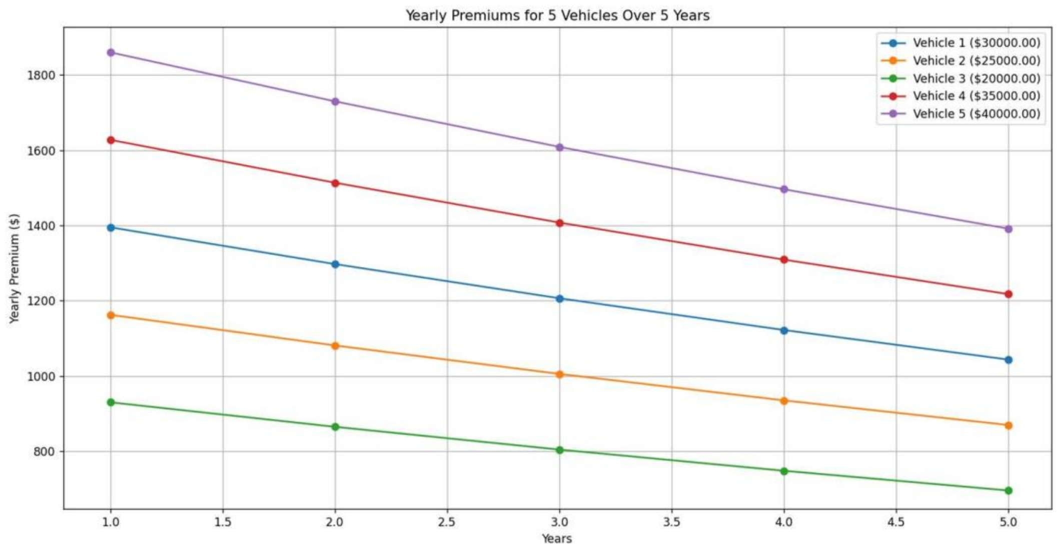
Yearly Premium: $1391.35

Quarterly Premium: $347.84

Monthly Premium: $115.95

Output Graph:

The program will also display a chart showing how each vehicle's yearly premiums decrease over time as the vehicle value depreciates. Each vehicle is represented by a unique line on the chart,making it easy to compare the premium trends across different vehicles.


Yearly Premiums for 5 Vehicles Over 5 Years
Vehicle 1 ($30000.00)
Vehicle 2 ($25000.00)
Vehicle 3 ($20000.00)
Vehicle 4 ($35000.00)
Vehicle 5 ($40000.00)
Yearly Premium ($)
Years