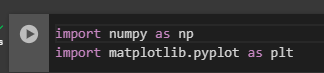
Name: Ashutosh Shrivastava Subject: Generative AI (CA2)

PRN: 21070521020 Section: A(A1)

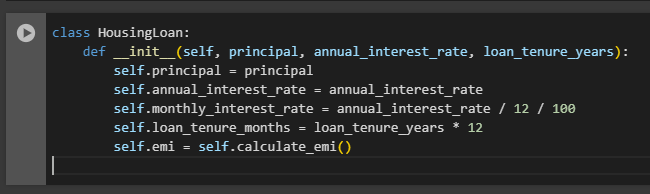
**Problem Statement:** Generate a model in Python to represent a Housing loan scheme and create a chart to display the Emi based on rate of interest and reducing balance for a given period. If a customer wishes to close the loan earlier, print the interest lost distributed over the remaining no. Of months. Assume suitable data and inputs as necessary.

**1. Import Libraries**



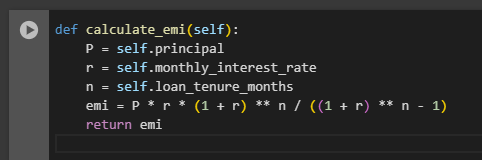
* **numpy:** A library for numerical operations, used to generate an array of interest rates for charting.
* **matplotlib**: A library for plotting graphs, used to visualize EMI values across different interest rates.

2. **Define the Housing Loan Class**



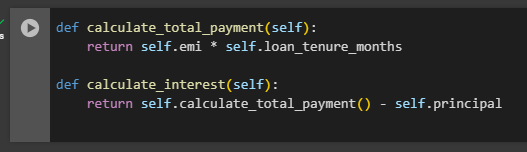
* **Defines the class Housing Loan**: Represents the housing loan with attributes like the loan amount, interest rate, and tenure.
* **Constructor**: Initializes the loan parameters:
  + principal (loan amount)
  + annual\_interest\_rate (annual interest)
  + Converts the annual interest rate to a monthly rate.
  + Converts the loan tenure from years to months.
  + Automatically calculates the EMI upon initialization using the calculate\_emi() method.

3. **Calculate EMI**



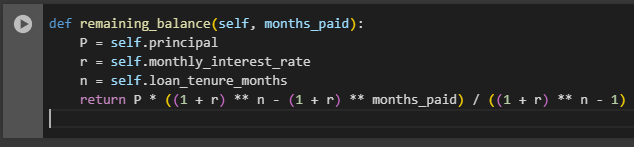
* **calculate\_emi() method**:
  + Implements the EMI formula to calculate the monthly installment (emi).
  + Takes into account the principal (P), monthly interest rate (r), and loan tenure in months (n).

4. **Total Payment and Interest Calculation**



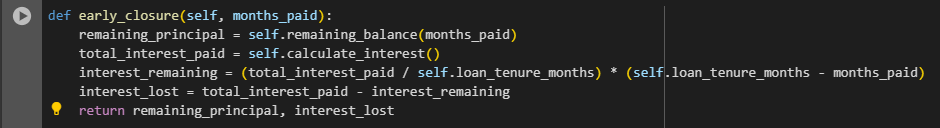
* **calculate\_total\_payment() method**:
  + Calculates the total amount that will be repaid over the loan period by multiplying the EMI with the total number of months.
* **calculate\_interest() method**:
  + Calculates the total interest paid over the entire loan period by subtracting the original principal from the total repayment amount.

5. **Calculate Remaining Loan Balance**



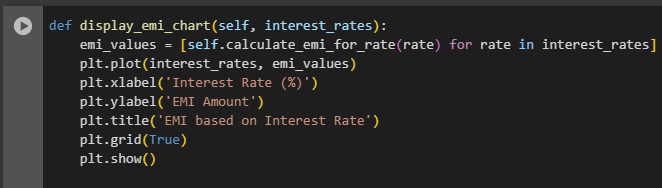
* **remaining\_balance() method**:
* Computes the remaining loan balance after a given number of months (months\_paid).
* Uses the reducing balance method to calculate the outstanding loan amount at any point in time.

6. **Early Closure and Interest Lost Calculation**



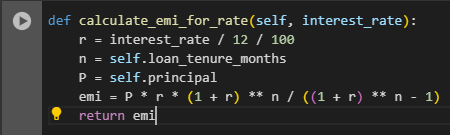
* **early\_closure() method**:
* Handles early loan closure.
* Calculates the remaining principal after a specific number of months (months\_paid).
* Calculates how much interest would have been paid for the remaining period.
* Computes the interest lost due to early closure by subtracting the remaining interest from the total interest payable.

7. **EMI Chart for Interest Rate Variations**

****

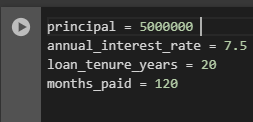
* **display\_emi\_chart() method:**
* Accepts a range of interest rates.
* Plots a chart showing how EMI changes for different interest rates using matplotlib.
* Helps visualize the effect of varying interest rates on the EMI.

**8. Calculate EMI for Different Interest Rates**

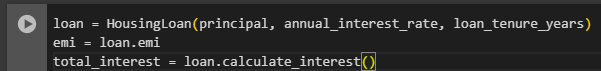
****

* **calculate\_emi\_for\_rate() method:**
* Calculates EMI for a given interest rate (other than the default one for the loan).
* This is used when generating the EMI chart with varying interest rates.

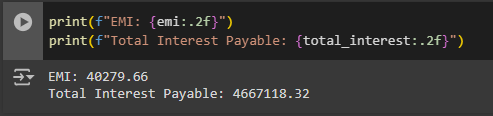
**9. Initializing sample Parameters**

****

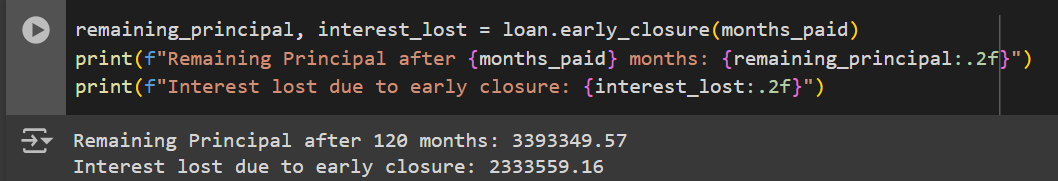
**10. Initializing Housing loan object**

****

**11. Printing EMI and Total Interest**

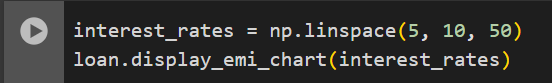
****

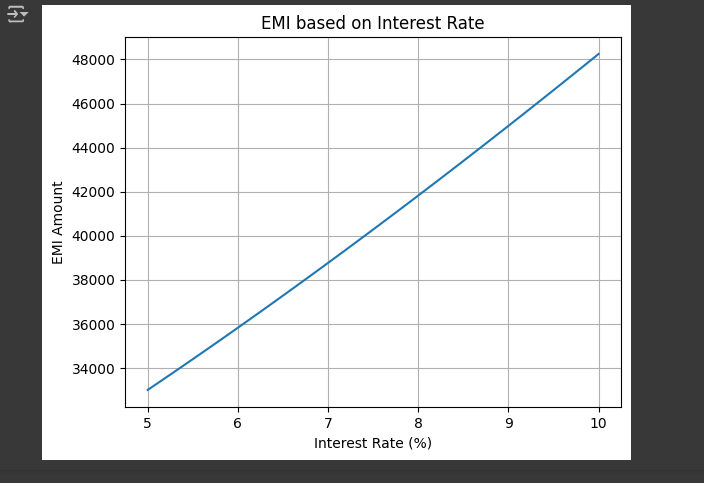
**12. Early closure calculation**

****

* Calculates amd prints the remaining principal and interest lost if the loan is closed early after 120 months

**13. Visualizing results**

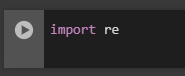
****

****

* Plots an EMI chart for interest rates between 5% and 10%

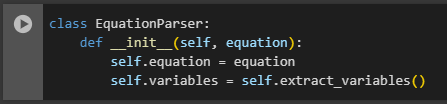
**Problem Statement:** Generate a model to represent a mathematical equation, write a program to parse the equation, and ask for input for each parameter.

1. **Imports and Regular Expressions for Parsing**



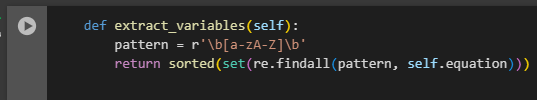
* **re**: This imports the re module for working with regular expressions.
* Regular expressions allow us to search for specific patterns in the equation string (like variable names).

2. **Equation Parser Class Initialization**



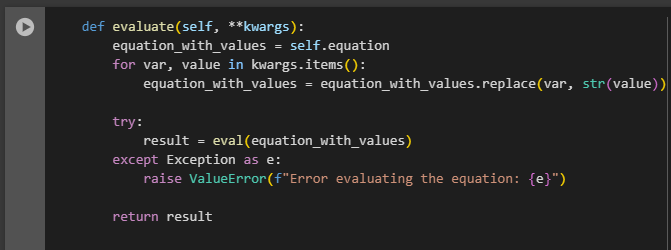
* **class EquationParser**
  + Defines the EquationParser class, which encapsulates the logic for handling a mathematical equation.
* **\_\_init\_\_(self, equation)**:
  + This is the constructor method. It initializes the object with the equation provided by the user.
* **self.equation**
  + Stores the mathematical equation as a string.
* **self.variables**
  + Calls the extract\_variables method to get a list of variables (parameters) from the equation.

3. **Extracting Variables Using Regular Expressions**



* **extract\_variables(self)**: This method extracts the variables from the equation.
  + **pattern = r'\b[a-zA-Z]\b'**: The regular expression pattern that matches single alphabetic characters (like a, b, x), which are typically used as variables in mathematical equations.
  + **re.findall(pattern, self.equation)**: Finds all occurrences of individual alphabetic characters in the equation.
  + **set(re.findall(...))**: Ensures that duplicate variables are removed (since a variable might appear more than once in the equation).
  + **sorted(set(...))**: Returns a sorted list of unique variables.

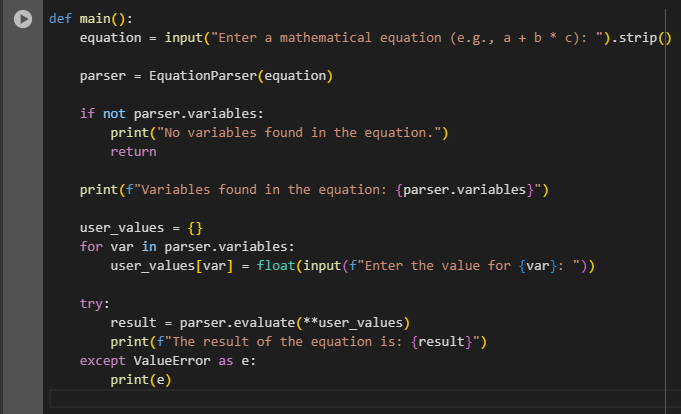
4. **Evaluating the Equation with User Input**



**evaluate(self, \*\*kwargs)**: This method evaluates the equation by replacing the variables with the corresponding values provided by the user.

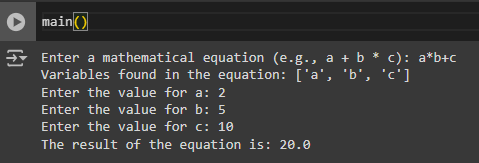
* **kwargs.items()**: Contains key-value pairs where the key is the variable (e.g., a, b) and the value is the user input.
* **equation\_with\_values = equation\_with\_values.replace(var, str(value))**: Replaces each variable in the equation with the corresponding user input value.
* **eval()**: Safely evaluates the equation string with the substituted values.
* **try...except**: Handles any potential errors that might occur during the evaluation (e.g., invalid equation format).

5. **Main Program to Get Input and Evaluate the Equation**



* **main()**: This is the main function that interacts with the user and calls the EquationParser class methods.
  + **input()**: Prompts the user to enter a mathematical equation.
  + **parser = EquationParser(equation)**: Instantiates the EquationParser class with the equation provided by the user.
  + **parser.variables**: Contains the list of variables extracted from the equation.
  + **if not parser.variables**: Checks if any variables were found in the equation. If none, it prints a message and returns.
  + **for var in parser.variables**: Loops over each variable and asks the user to input a value for it.
  + **parser.evaluate(\*\*user\_values)**: Evaluates the equation by passing the user input values for each variable.
  + **try...except**: Catches any errors that might occur during the evaluation.

**6. Execution**

****