**NAME : CHADA SATHWIKA REDDY**

**HTNO:2403A51334**

**BATCH:14**

**LAB 6.1 AI BASED CODE COMPLETION**

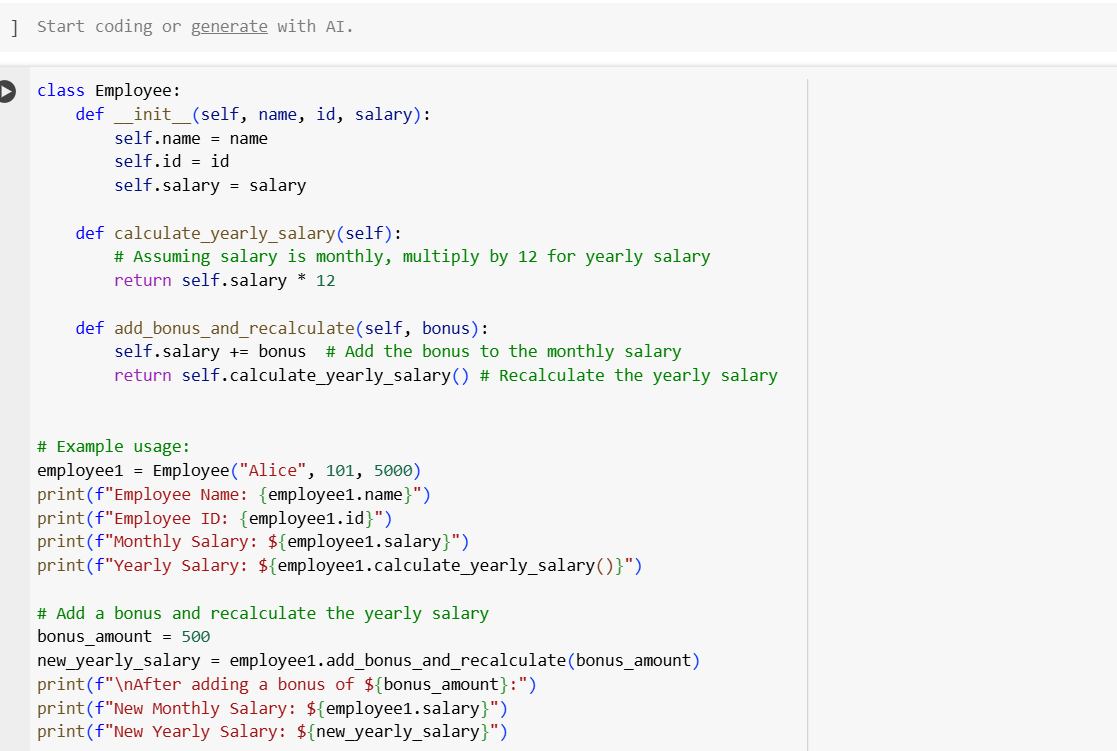
**TASK1**

**PROMPT:**write a python program that creates an employee class with certain attributes

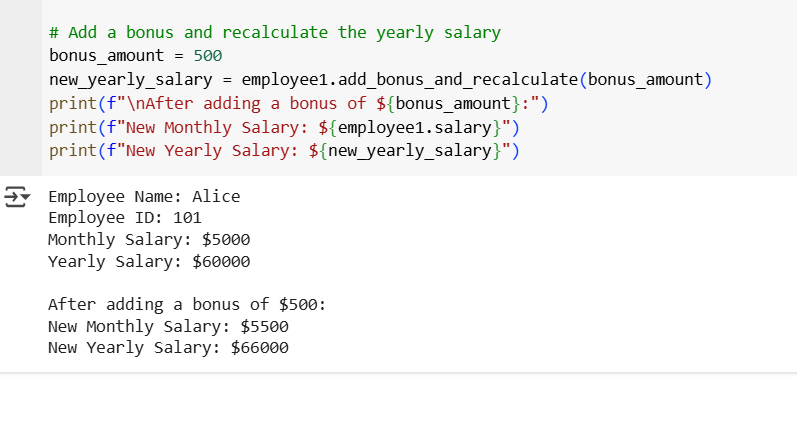
Method to calculate annual income

Method to add bonus and recalculate annual income

**CODE**:



Output:

code

**code explanation:** def calculate\_yearly\_salary(self): This method calculates the yearly salary of an employee.

* self: Refers to the instance of the class.
* It assumes the salary attribute stores the monthly salary.
* return self.salary \* 12: It multiplies the monthly salary by 12 to get the yearly salary and returns the result.

def add\_bonus\_and\_recalculate(self, bonus): This method adds a bonus to the employee's monthly salary and then recalculates the yearly salary.

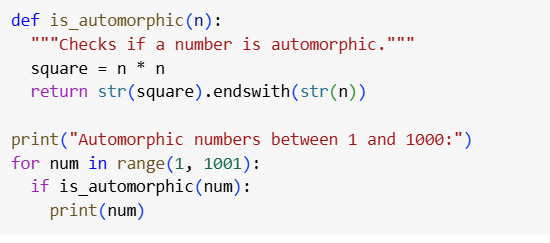
* self: Refers to the instance of the class.
* bonus: This parameter is the amount of bonus to be added.
* self.salary += bonus: This line adds the bonus amount to the existing self.salary (monthly salary).
* return self.calculate\_yearly\_salary(): After adding the bonus, it calls the calculate\_yearly\_salary method to get the new yearly salary and returns it.

**TASK2:**

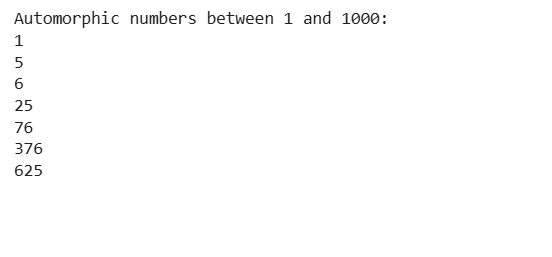
**PROMPT:** Write a python program that prints automorphic numbers between 1 to 1000 using for loop

**CODE:**

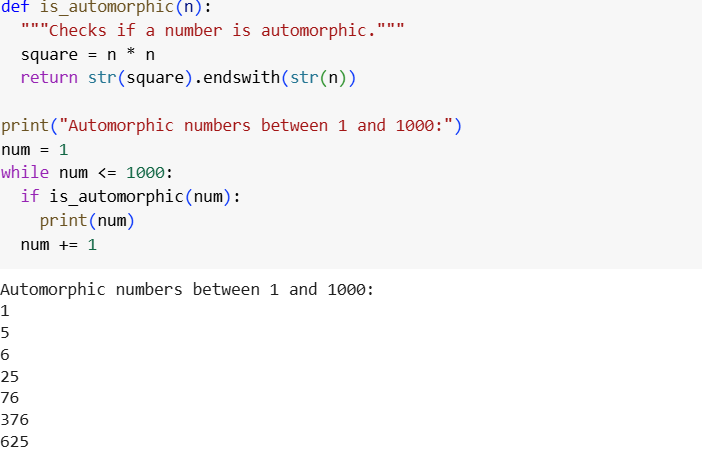
Using for loop:



Output:



Using while loop:



Comparision between both implementations:

| * **Feature** | * **For Loop** | * **While Loop** |
| --- | --- | --- |
| * Loop Control | * Uses range() — clean & concise | * Manual control via counter |
| * Readability | * More readable in this case | * Slightly more verbose |
| * Flexibility | * Less flexible for dynamic steps | * More flexible for varying logic |
| * Performance (1–1000) | * Equal performance | * Equal performance |
| * Use Case | * Best when number of iterations is known | * Best when iterations depend on conditions |

Code observation:

Both implementations are correct and efficient.

They produce the same results.

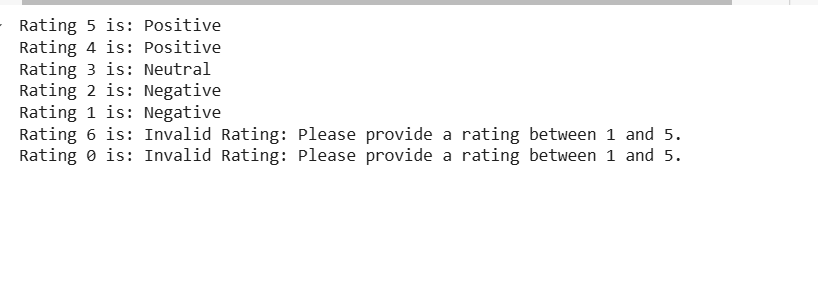
The for loop is preferable in readability and simplicity for this problem.

TASK3:

PROMPT: Write a Python function using **nested if-elif-else statements** that takes a numerical feedback rating (1 to 5) and classifies it as:

* "Positive" for ratings 4 and 5
* "Neutral" for rating 3
* "Negative" for ratings 1 and 2:
* CODE: 

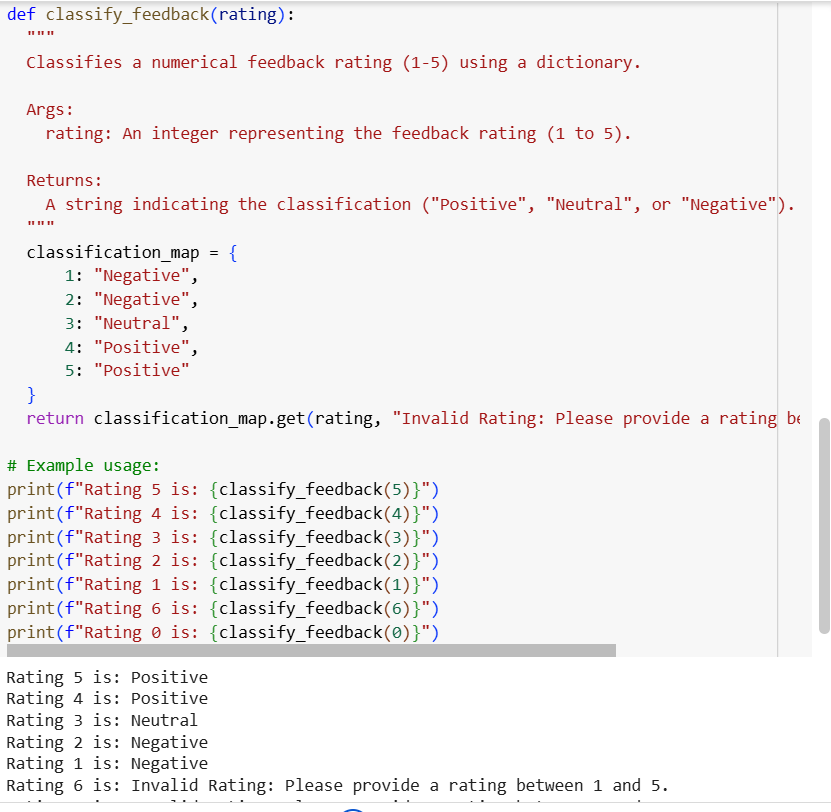
OUTPUT:



Using match-case:



Using dictionary:



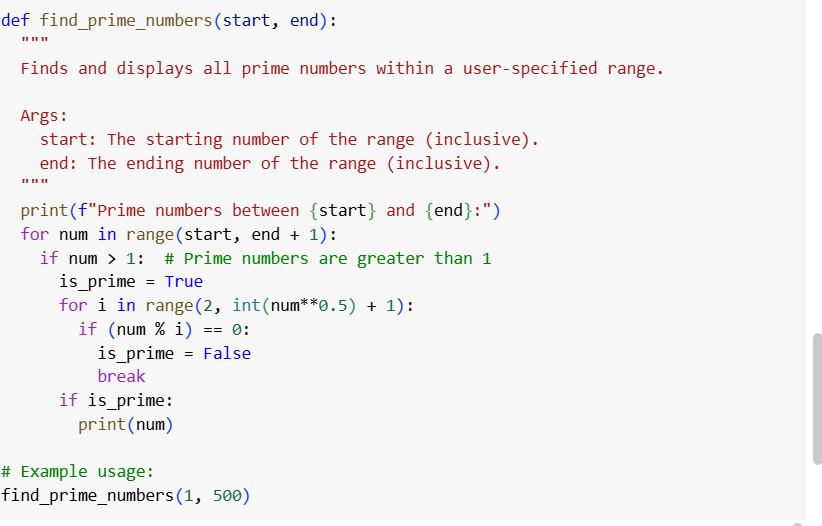
Comparision between implementations

| **Feature** | **Nested if-elif-else** | **match-case** | **Dictionary-based** |
| --- | --- | --- | --- |
| Readability | Moderate | Very clear | Very concise |
| Python Version | Any | Python 3.10+ | Any |
| Extendability | Easy | Easy | Easiest |
| Performance | Slightly slower | Fast | Fastest (direct lookup) |
| Use Case Fit | | Good for logic-based rules | Good for value matching | Best for value-to-label maps |

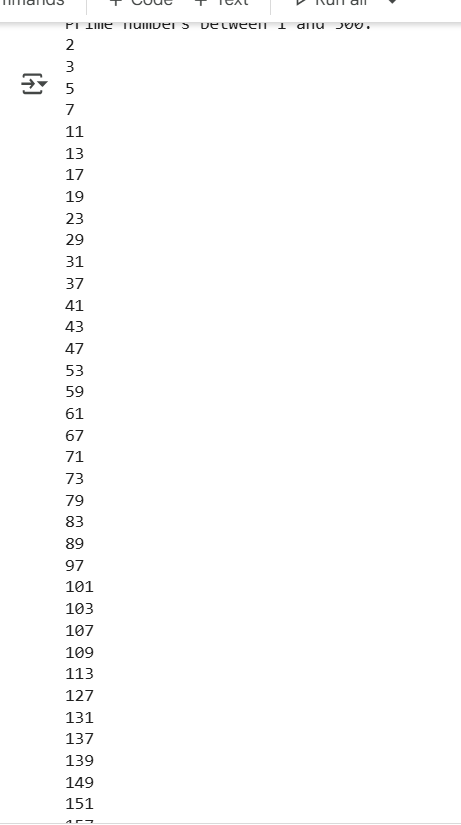
**TASK 4:**

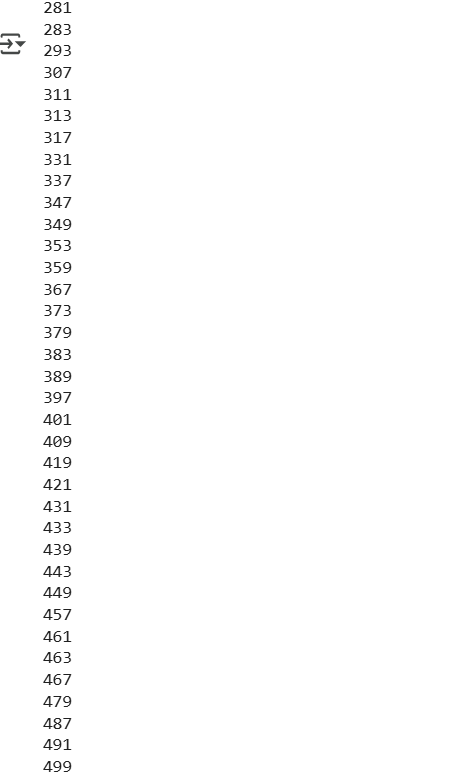
**PROMPT:**Generate a function using AI that displays all prime  
numbers within a user-specified range 1 to 500

**CODE:**

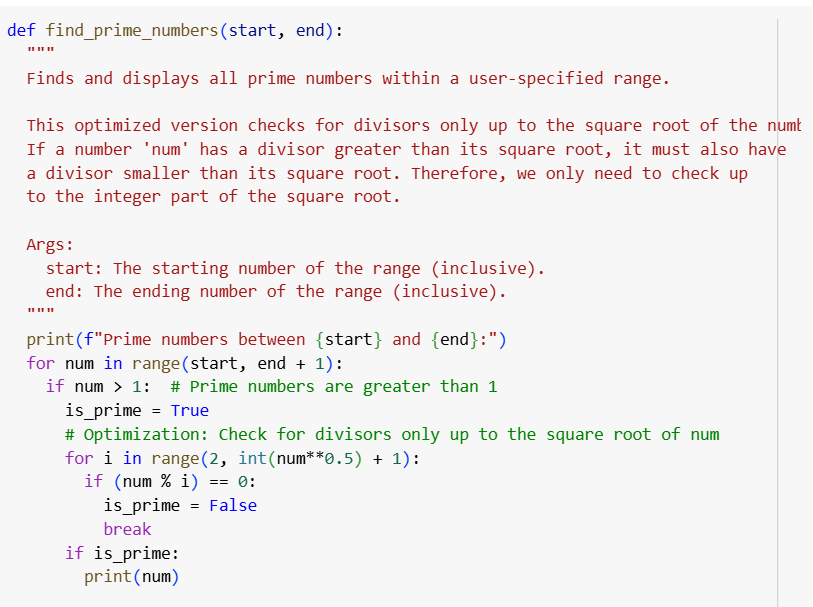
****

**OUTPUT:**

****

****

**OPTIMIZED CODE:**

****

**Analysis of Initial Code**

Correctness:

Correctly identifies primes by checking if the number is divisible by any number from 2 up to one less than itself.

Efficiency:

Inefficient because it checks all possible divisors, leading to O(n^2) time complexity for listing primes in the range.

Unnecessary to check divisibility beyond half the number, or better, beyond its square root.

 The optimized version reduces the number of divisibility checks to only those numbers from 2 up to the square root of num.

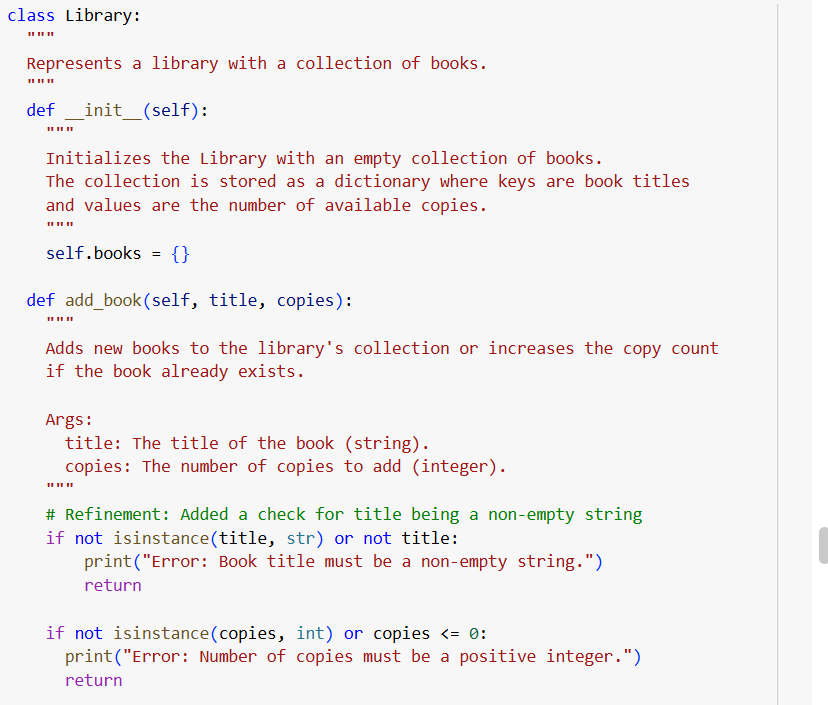
 This significantly reduces computation time because if num has a divisor larger than its square root, it must also have a smaller corresponding divisor.

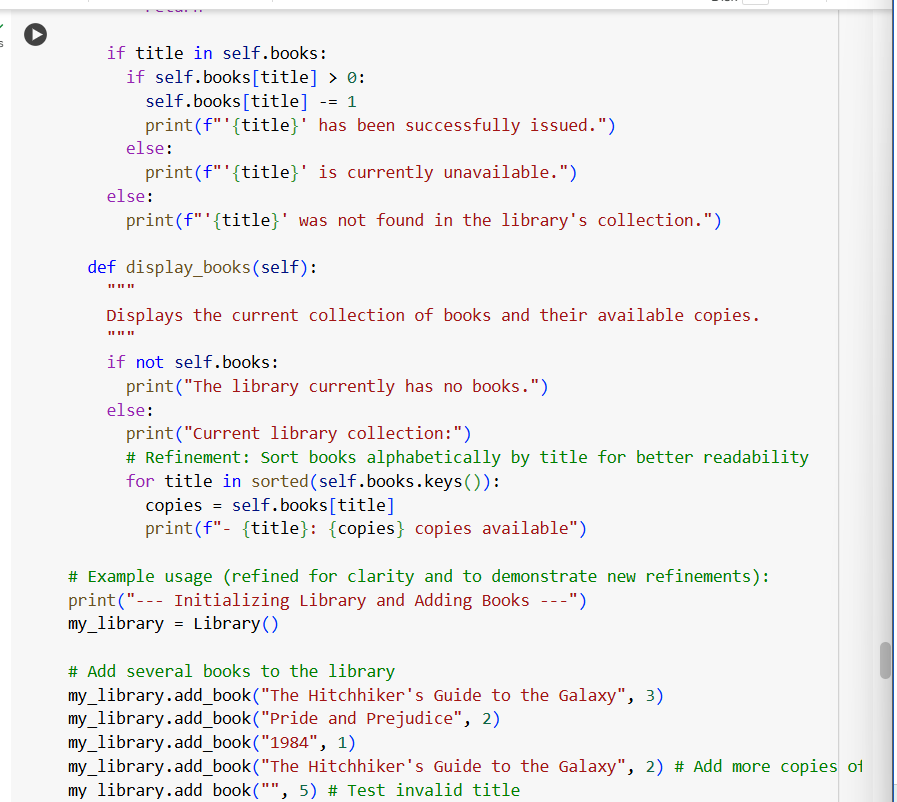
 This reduces the average time complexity closer to O(n√n) from O(n^2).

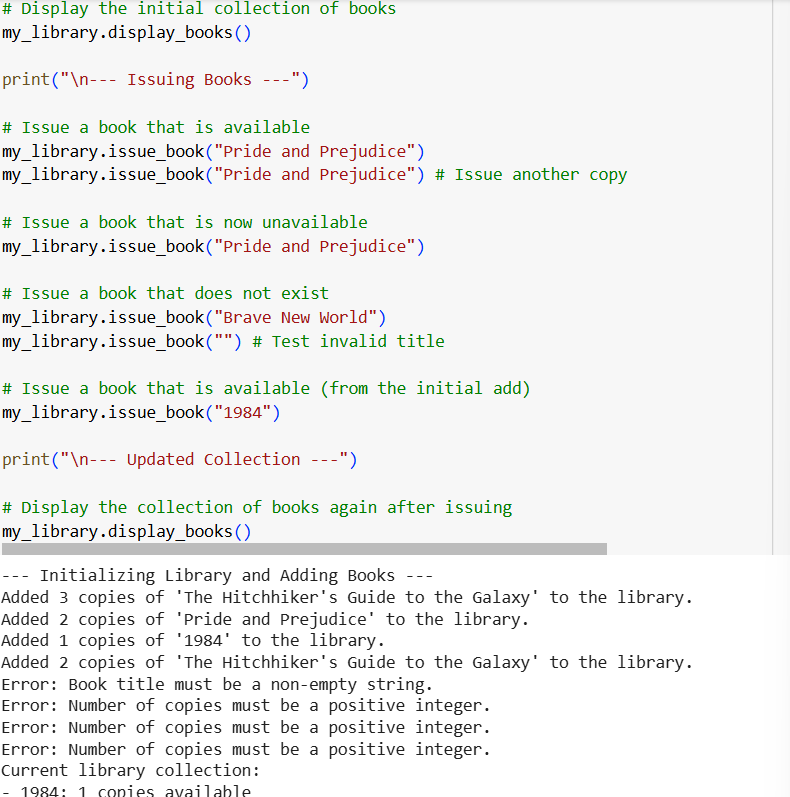
**TASK5:**

**PROMPT:** Write a python program that builds a Library class with methods to add\_book(), issue\_book(), and display\_books() ALso handle edge cases such as unvailability issuing and add comments and documentation

**CODE:**







**Explanation:**

* The Library class maintains a dictionary books where keys are book names, and values are booleans indicating availability.
* add\_book() adds or resets the book’s status to available.
* issue\_book() checks if the book exists and is available before issuing, handling cases where it’s missing or already issued.
* display\_books() lists only the books currently available to borrow.
* Edge cases are handled gracefully with appropriate messages.
* Inline comments and docstrings clarify the purpose and behavior of each method