**Name: B.Sathvika**

**HTNO:2403a51344**

**Batch:14**

**Lab 6.1:** **AI-Based Code Completion – Classes, Loops, and Conditionals**

**Task 1:**

**Prompt:** Create a Python class named Employee with the following attributes: name, id, and salary (monthly salary).

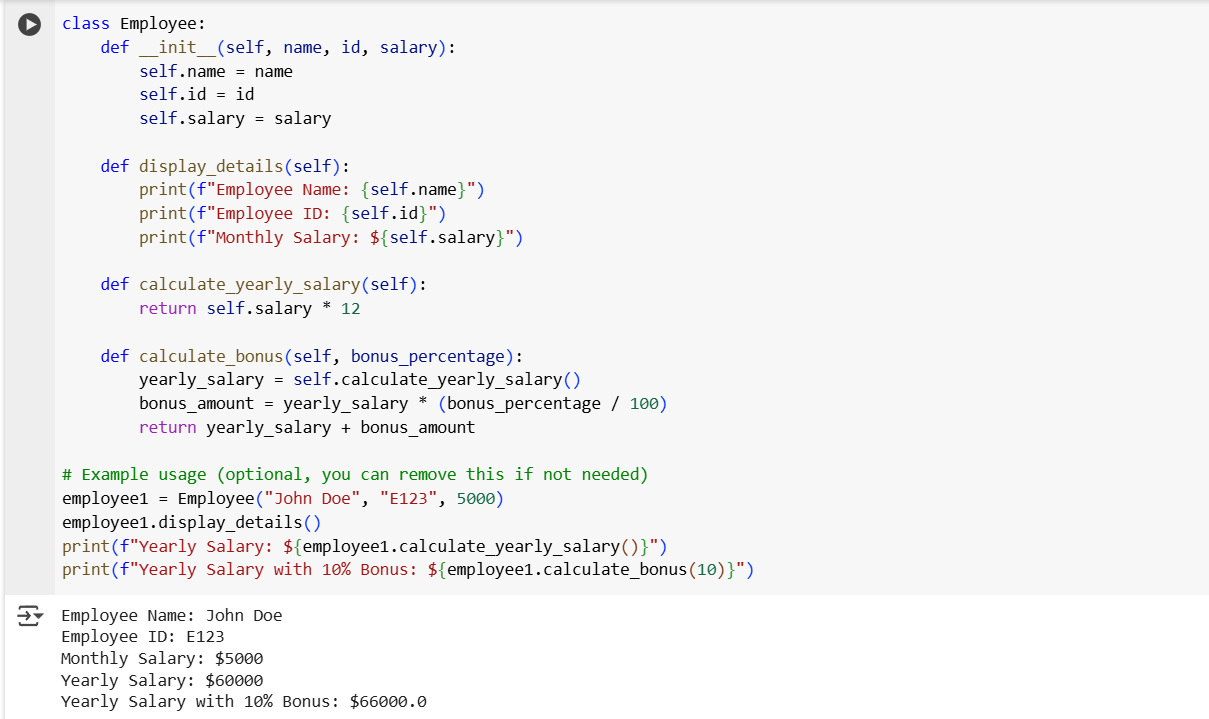
Add a method display\_details() to print the employee’s name, id, and monthly salary.

Add another method calculate\_yearly\_salary() that returns the yearly salary by multiplying the monthly salary by 12.

After that, add a method calculate\_bonus(bonus\_percentage) which calculates the yearly salary including a given bonus percentage.

The class should be well-structured and properly use the constructor.

**Code and Output:**

****

**Observation:**

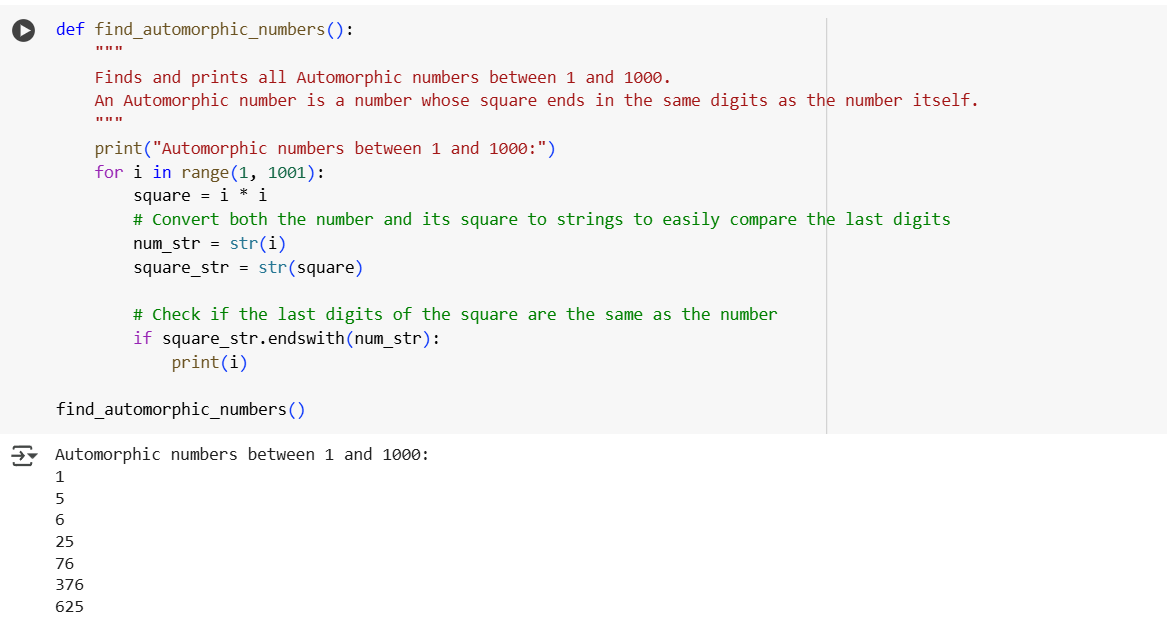
1. Constructor (\_\_init\_\_)  
   Correctly initializes name, emp\_id, and salary.
2. display\_details() Method  
   Properly displays employee details.
3. calculate\_yearly\_salary() Method  
   Correctly calculates yearly salary by multiplying monthly salary by 12.

The structure looks good, well-organized, and correct.

**Task2:**

**Prompt1:** Write a Python function that finds and prints all Automorphic numbers between 1 and 1000 using a for loop.

**Code and Output:**

****

**Analyzation of the correctness and efficiency of the generated logic.**

**Correctness:**

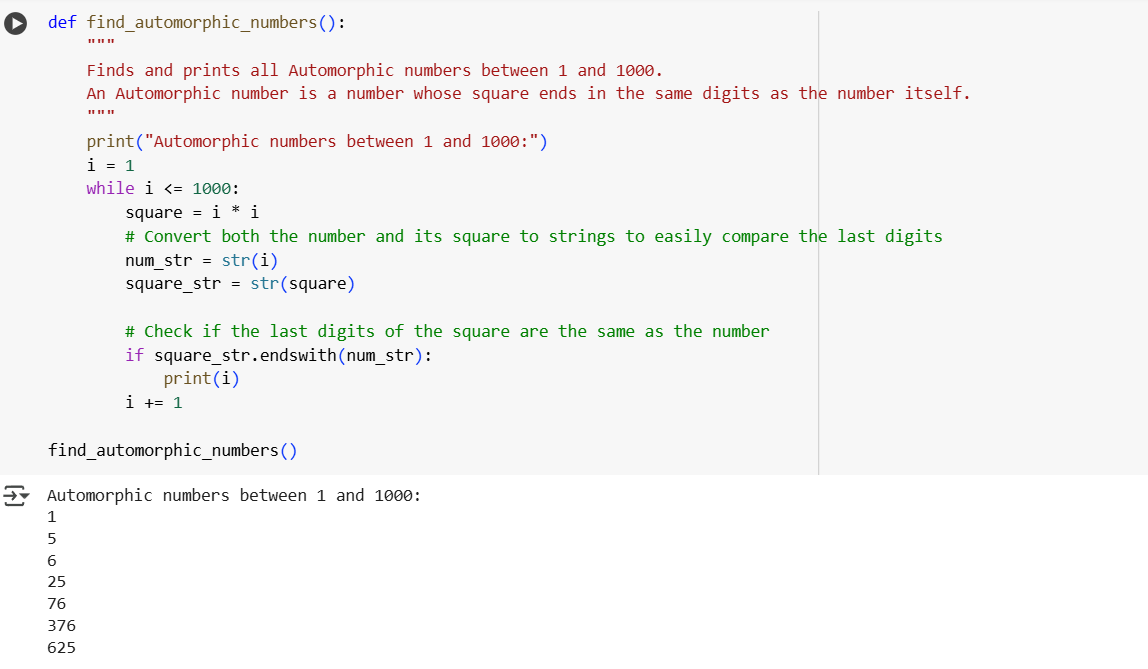
* The code correctly iterates from 1 to 1000.
* For each number, it calculates its square and checks if the square ends with the number itself using str(square).endswith(str(num)).
* This is correct and simple logic.

**Efficiency:**

* Time Complexity: O(n \* m), where n = 1000 and m is the length of the string comparison.
* For small ranges like 1 to 1000, this is acceptable and works well.

**Prompt2:** Regenerate the function to list Automorphic numbers between 1 and 1000 using a while loop instead of a for loop.

**Code and Output:**

****

**Comparing of Both Implementations**

| **Feature** | **For Loop Implementation** | **While Loop Implementation** |
| --- | --- | --- |
| Readability | Simple and clear | Simple and clear |
| Efficiency | Same for this problem | Same for this problem |
| Initialization | Handled by the for loop | Requires explicit initialization of num = 1 |
| Increment | Automatic in for loop | Manual increment num += 1 required |
| Use Case | Preferred when range is known | Useful when loop depends on external conditions |

**Task3:**

**Prompt1:** Write Python code using nested if-elif-else statements that classifies online shopping feedback based on a numerical rating (1 to 5).

* 4 or 5 → Positive
* 3 → Neutral
* 1 or 2 → Negative

**Code and Output:**

****

**Analyzation of correctness and readability**

**Correctness:**

* Properly checks if rating is between 1 and 5.
* Classifies ratings correctly.

**Readability:**

* Readable, but slightly verbose due to nested structure.
* A simple flat structure would be cleaner**.**

**Prompt2:** Rewrite the feedback classification function using either a dictionary-based approach or a match-case structure for better clarity and efficiency**.**

**Code and Output:**

****

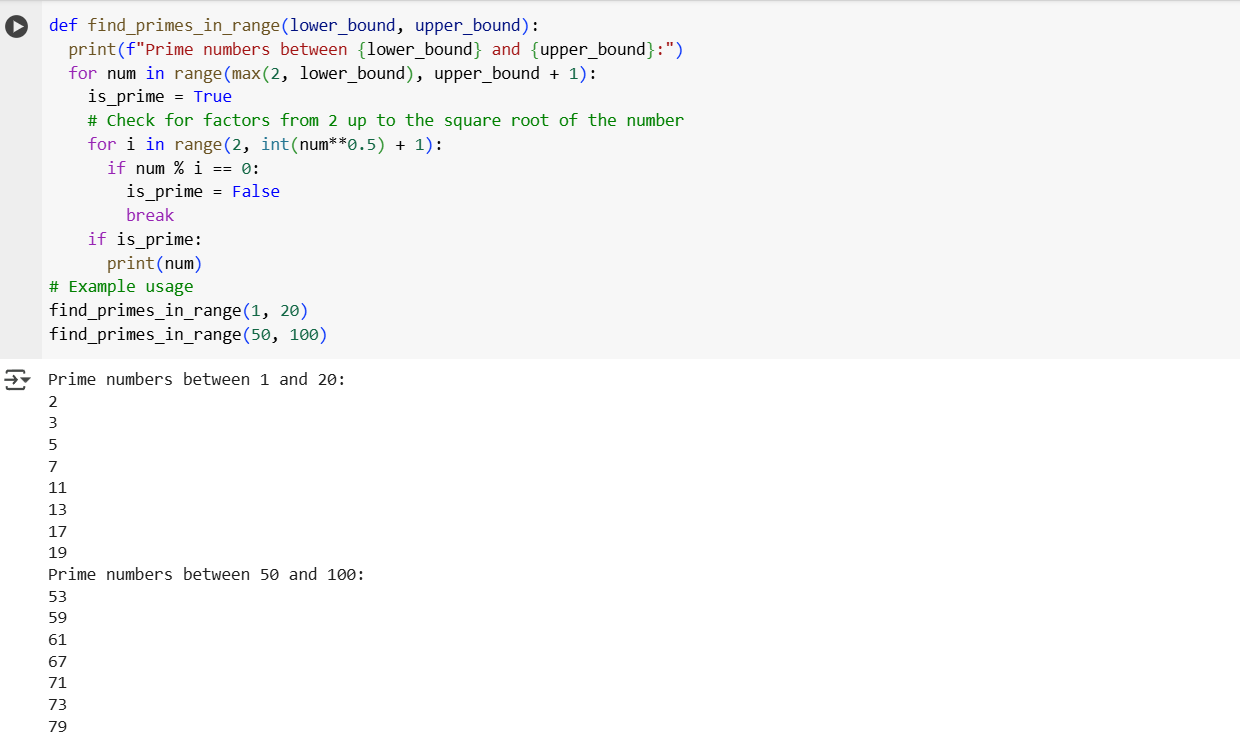
**Comparison of Implementations**

| **Feature** | **If-Elif-Else Implementation** | **Match-Case Implementation** | **Dictionary Implementation** |
| --- | --- | --- | --- |
| Readability | Good but slightly verbose | Very clean and modern | Very concise and efficient |
| Python Version Support | All versions | Requires Python 3.10+ | All versions |
| Efficiency | Fine for small range | Fine and readable | Fast lookup with .get() |
| Scalability | Less scalable for large mappings | Easy to extend | Very easy to extend |

**Task4:**

**Prompt1:** Write a Python function that takes a lower and upper bound as input and prints all prime numbers in that range using a for loop and simple prime-checking logic.

**Code and Output:**

****

**Analyzation of correctness and efficiency**

**Correctness:**

* The code correctly loops from lower to upper.
* For each number >1, it checks divisibility from 2 to (num - 1).

**Efficiency:**

* Time Complexity: O(n²) in the worst case → not efficient for large ranges.
* Checking all numbers up to num - 1 is wasteful**.**

**Prompt2:** Regenerate the prime number listing function but optimize the prime-checking logic by testing divisibility only up to the square root of the number

**Code and Output:  
**

**Comparison of Both Implementations**

| **Feature** | **Simple Version** | **Optimized Version** |
| --- | --- | --- |
| Time Complexity | O(n²) | O(n \* √n) |
| Readability | Easy to understand | Still easy to understand |
| Performance (large ranges) | Slow | Much faster |
| Best Practice | Not recommended | Recommended for practical use |

**Task5:**

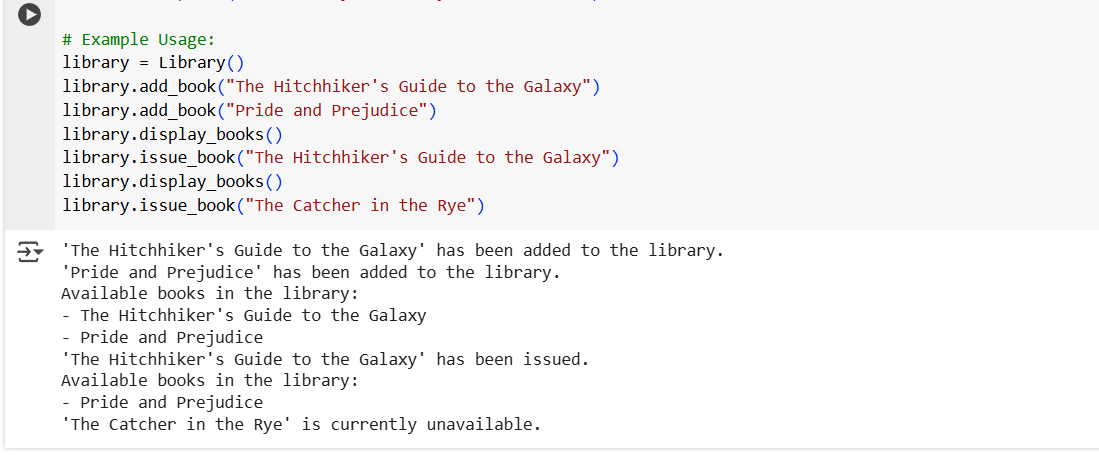
**Prompt:** Write a Python class named Library with the following methods:

* add\_book(book\_name): Adds a book to the library collection.
* issue\_book(book\_name): Issues a book if available, otherwise prints a message that the book is unavailable.
* display\_books(): Displays all available books in the library.

Add inline comments and a class docstring to the Library class for clarity and better understanding.

**Code and Output:**

****

****

**Analyzation of correctness and edge case handling**

**Correctness:**

* add\_book(): Adds book to the list → correct.
* issue\_book(): Checks if the book exists before removing → handles unavailable books well.
* display\_books(): Displays all books or shows "No books available".

**Edge Cases:**

* Handles trying to issue a book not in the list by printing a clear message → Proper handling.
* Allows multiple additions of the same book → Could be fine or may require extra logic depending on requirements.