**LAB ASSIGNMENT-6.1**

**Name:**P.Navya

**HTNO:**2403A51331

**Batch:**13

**Task:1**

Create an Employee class with attributes (name,id, salary) and a method to calculate yearly salary

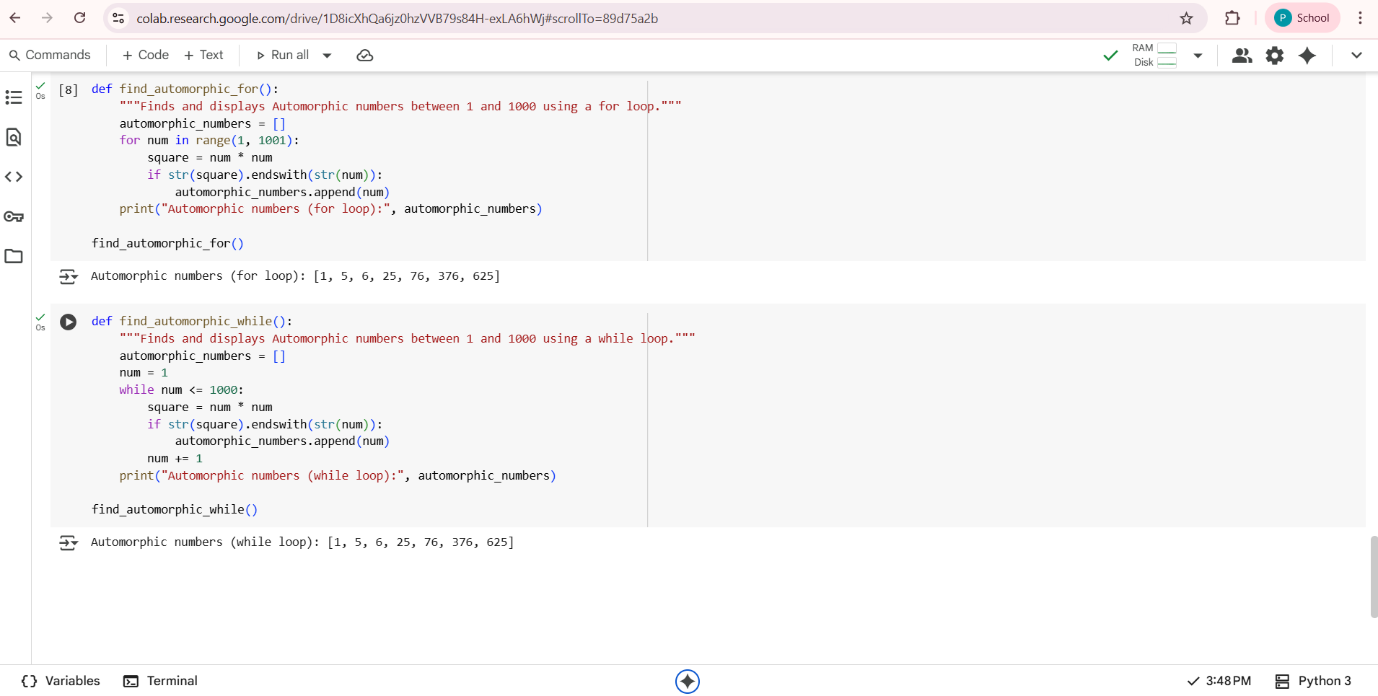


Explanation:

* **class Employee:**: This line defines a new class named Employee.
* **\_\_init\_\_(self, name, id, salary):**: This is the constructor method. It's called when you create a new Employee object. It initializes the object's attributes: name, id, and salary. self refers to the instance of the class.
* **display\_details(self):**: This method prints the details of the employee (name, ID, and salary) to the console.
* **calculate\_yearly\_salary(self):**: This method calculates and returns the yearly salary by multiplying the monthly salary by 12.
* **give\_bonus(self, bonus\_amount):**: This method takes a bonus\_amount as an argument, adds it to the employee's current salary, and then prints a message indicating the bonus was given and the new salary.

**Task-2:**

Generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop and Regenerate using a while loop and compare both implementations



Comparision:

| **Feature** | **find\_automorphic\_for()** | **find\_automorphic\_while()** |
| --- | --- | --- |
| **Loop Type** | for loop with range() | while loop |
| **Initialization** | Handled automatically by range() | Manual (num = 1) |
| **Condition** | Handled automatically by range() | Explicit (while num <= 1000) |
| **Incrementing** | Handled automatically by for loop | Manual (num += 1) |
| **Conciseness** | Generally more concise for fixed ranges | Slightly more verbose for fixed ranges |
| **Flexibility** | Less flexible for dynamic conditions | More flexible for dynamic conditions |
| **Core Logic** | Identical (str(square).endswith(str(num))) | Identical (str(square).endswith(str(num))) |

**Explanation:**

**Loop Type**: One uses a for loop with range() for iterating through a sequence, while the other uses a while loop which continues as long as a condition is true.

**Initialization, Condition, and Incrementing**: The for loop handles these aspects automatically when using range(), making it more concise for fixed iterations. The while loop requires you to manually initialize a counter, define the continuation condition, and increment the counter within the loop.

**Conciseness:** For iterating over a known range, the for loop is generally more concise.

**Flexibility**: The while loop is more flexible when the number of iterations is not fixed beforehand or depends on a dynamic condition.

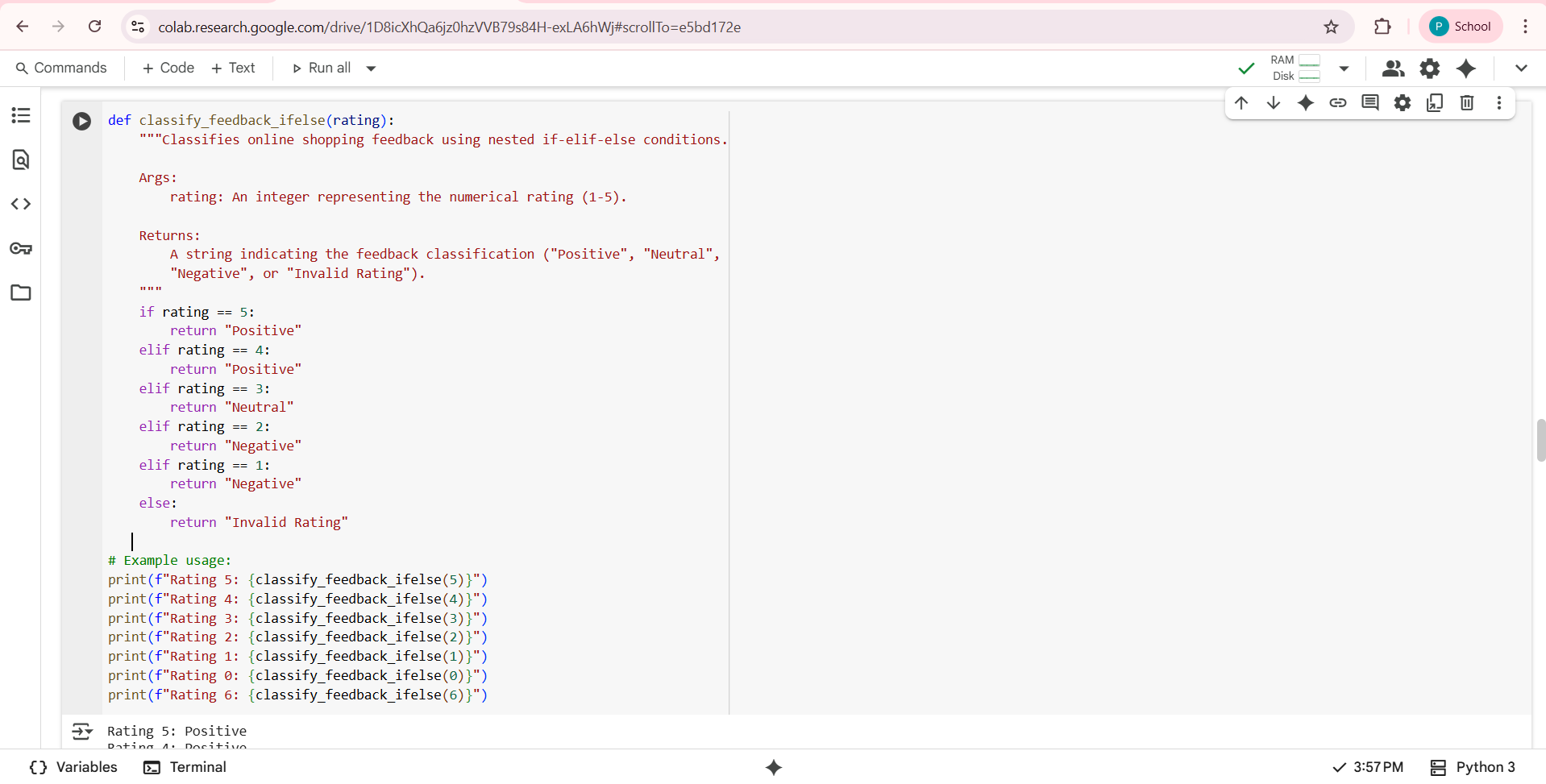
**Core Logic**: The actual code to check if a number is Automorphic is the same in both functions, demonstrating that the core problem-solving logic is independent of the looping mechanism.

**TASK-3:**

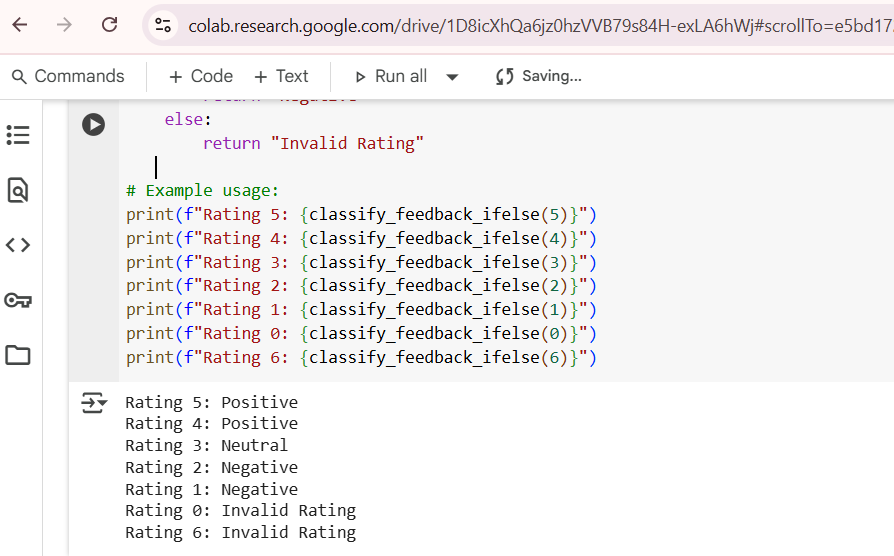
Write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral,or Negative based on a numerical rating (1–5) and

Rewrite using dictionary-based or match-case structure.

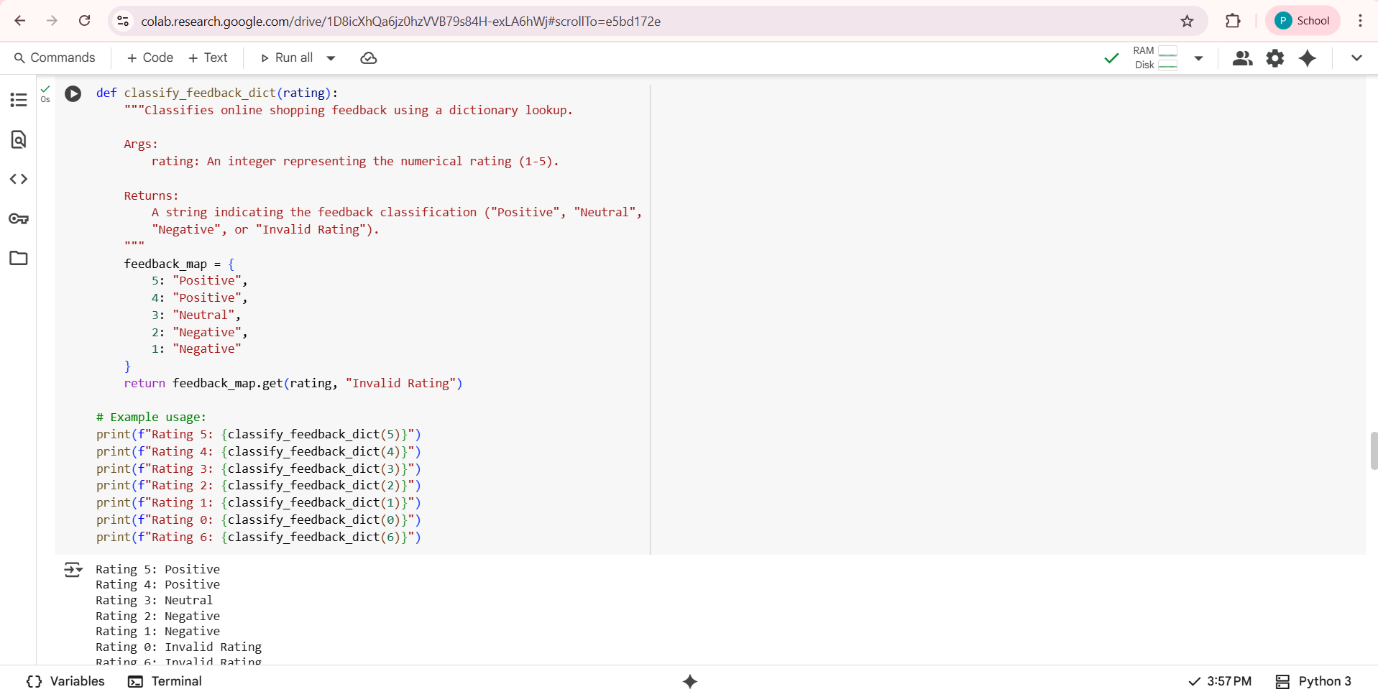
**Using Nested if-elif-else conditions:**

****

**Output:**

****

**Using dictionary-based:**

****

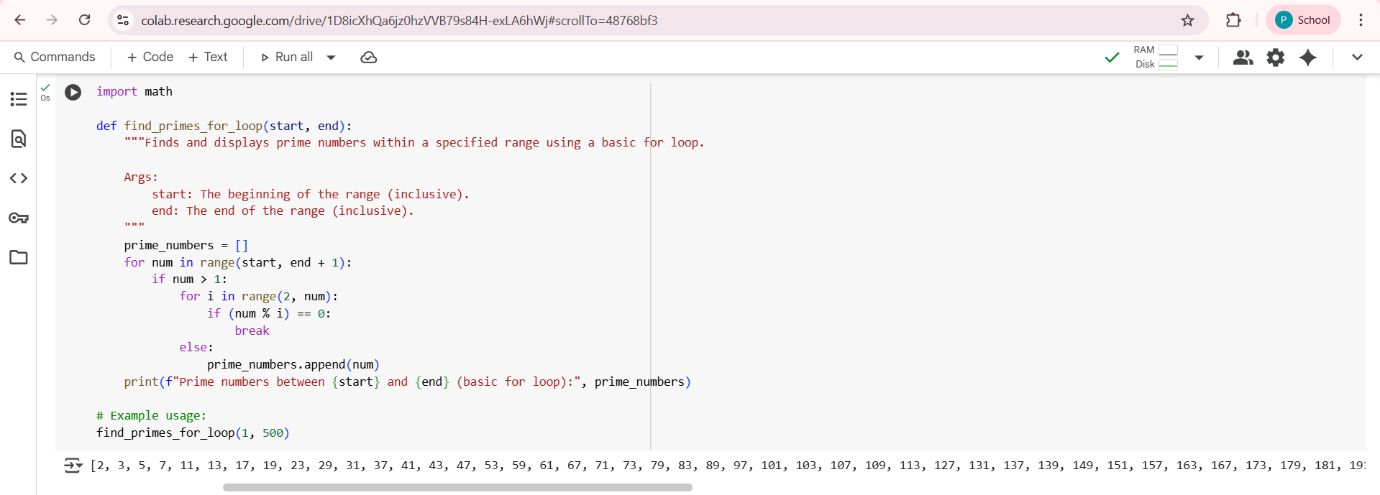
**Explanation:**

1. **Using nested if-elif-else statements:**This is a straightforward approach where each rating is checked sequentially using conditional statements. It's easy to understand for a few conditions but can become less readable as the number of conditions increases.
2. **Using a dictionary:**This approach maps each rating (key) directly to its corresponding classification (value). It's very readable and easy to maintain, especially when you have a direct mapping of inputs to outputs. It's also efficient for lookups.
3. **Using a match-case structure:**Available in Python 3.10+, this provides a more structured way to handle multiple distinct cases based on the value of an expression. It can be more readable than long if-elif-else chains and supports more advanced pattern matching.

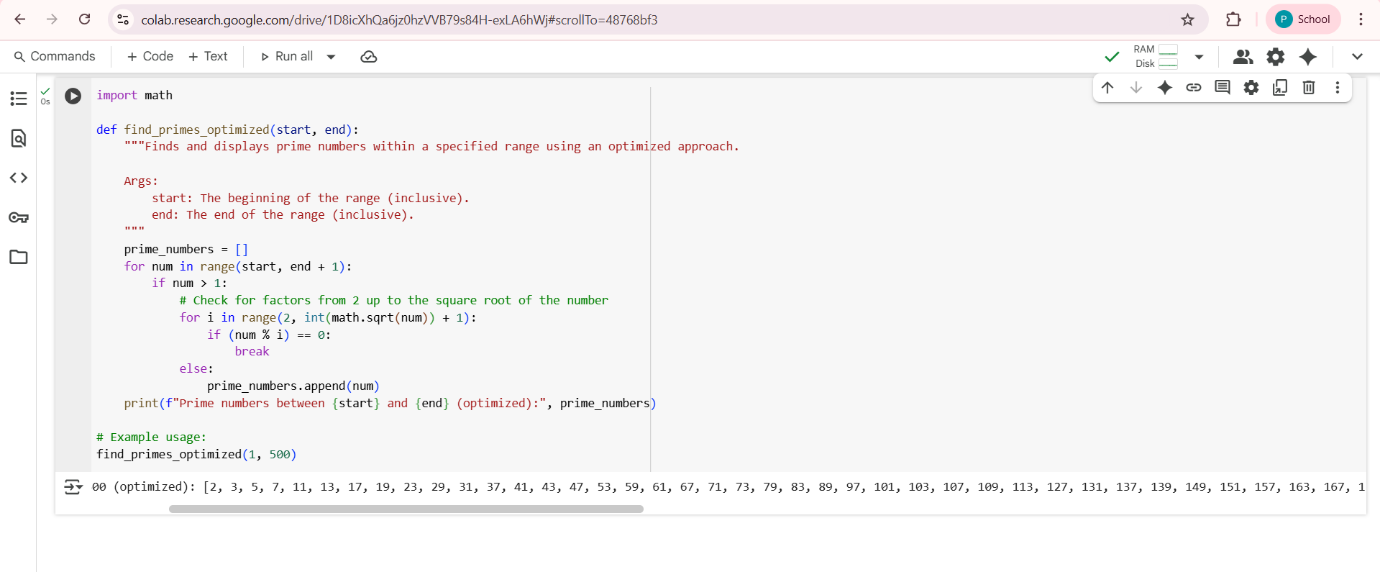
**Task-4:**

Generate a function that displays all prime numbers within a user-specified range (e.g., 1 to 500) .list all primes using a for loop and regenerate an optimized version (e.g., usingthe square root method

**Using for loop**:

****

**Optimized version of code:**

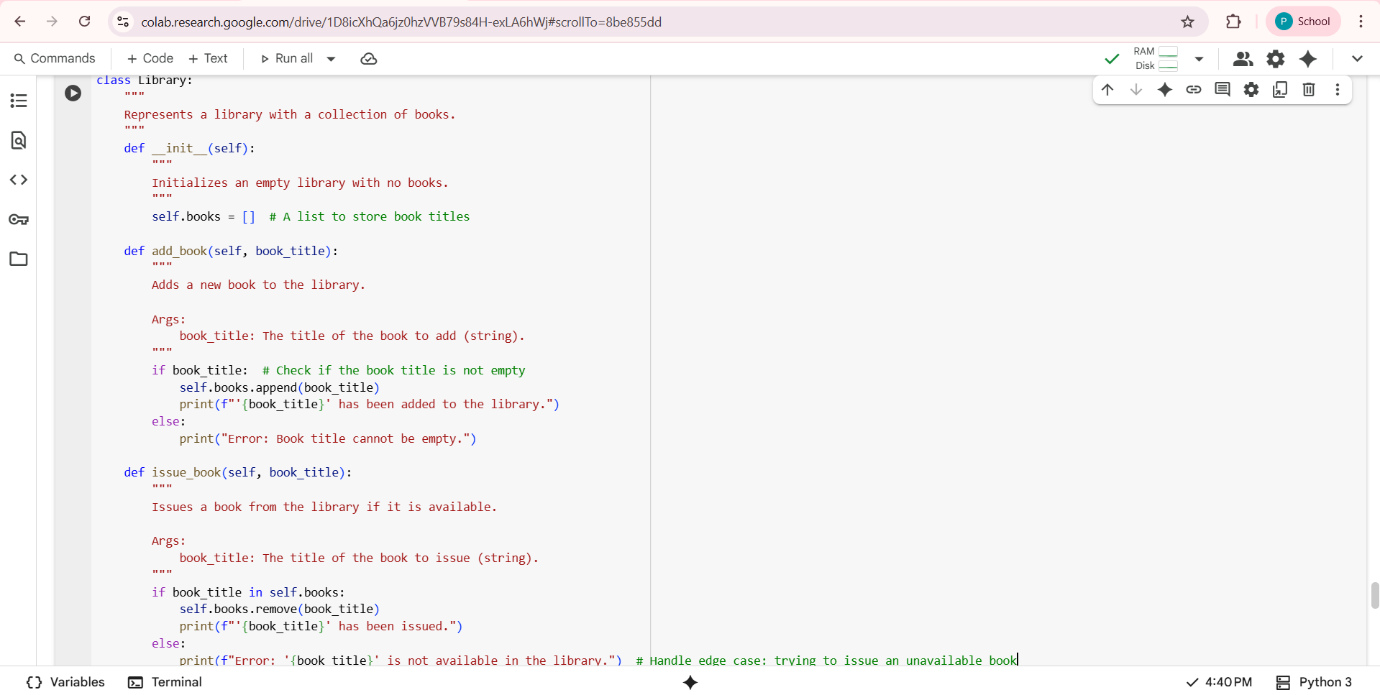
****

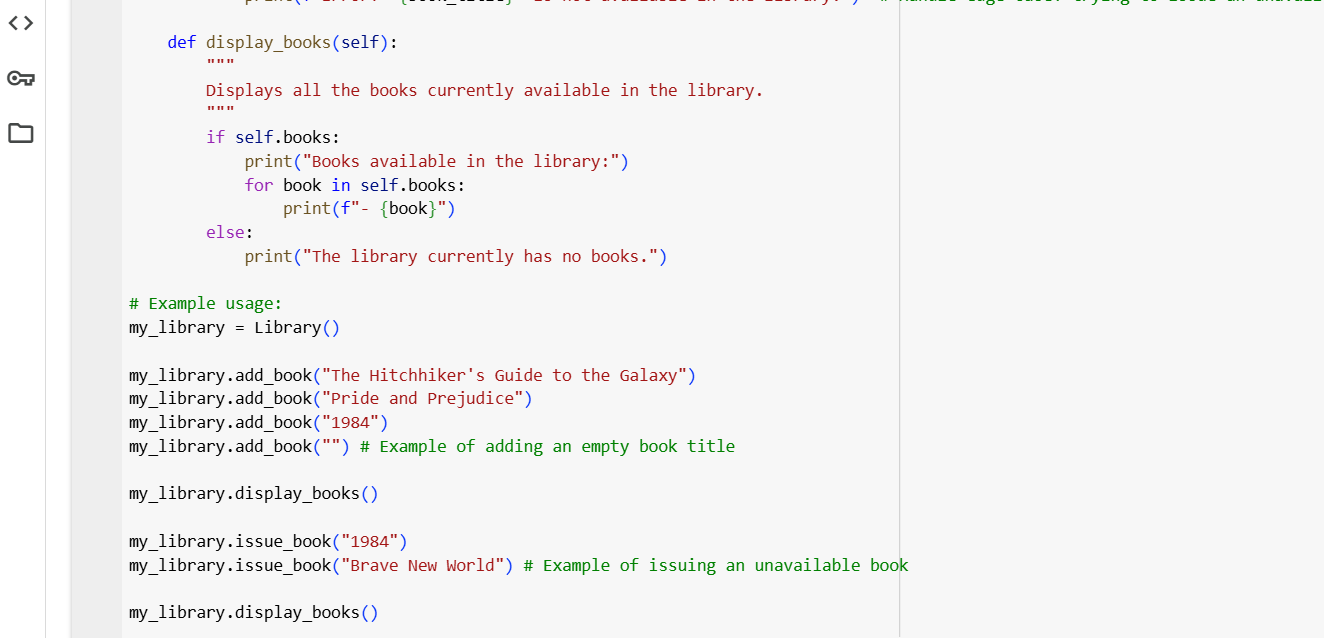
**Explanation:**

* **The Optimization:**A number num is prime if it is not divisible by any number other than 1 and itself. To check for primality, we traditionally check for divisibility by all numbers from 2 up to num - 1. If a number num has a divisor greater than its square root, it must also have a divisor smaller than its square root. Therefore, we only need to check for divisibility by numbers from 2 up to the integer part of the square root of num.
* **import math:** This line imports the math module, which provides mathematical functions like sqrt() for calculating the square root**.**
* **int(math.sqrt(num)) + 1:** Inside the inner for loop, this calculates the integer part of the square root of the current number (num) and adds 1 to include the square root itself in the range of numbers to check for divisibility.
* **for i in range(2, int(math.sqrt(num)) + 1)::** This loop iterates through potential divisors starting from 2 up to the calculated square root.
* **if (num % i) == 0::** If num is divisible by any number i in this range, it means num is not prime, so we break out of the inner loop**.**
* **else::** The else block associated with the inner for loop is executed only if the loop completes without finding any divisors (i.e., the break statement was not hit). This indicates that the number is prime, and it is appended to the prime\_numbers list**.**

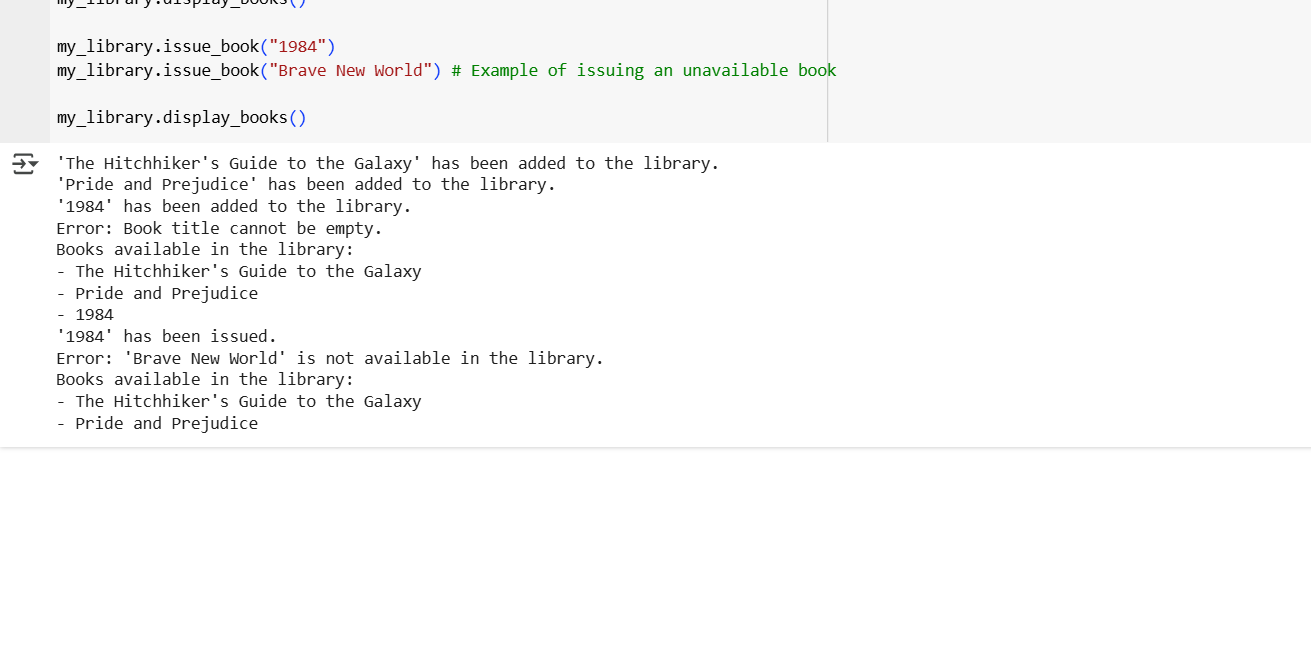
**Task-5:**

Use AI to build a Library class with methods to add\_book(), issue\_book(), and display\_books() include issuing unavailable books.  
o Ask AI to add comments and documentation





**Output:**

****

**Explanation:**

* **class Library::** This line declares a new class named Library.
* **"""Represents a library with a collection of books.""":** This is a docstring that explains the purpose of the class.
* **\_\_init\_\_(self)::** This is the constructor. It's called when you create a Library object. It initializes an empty list called self.books to store the titles of the books in the library.
* **add\_book(self, book\_title)::** This method is used to add a book to the library. It takes book\_title as an argument. It includes a check to ensure the book title is not empty before adding it to the self.books list and prints a confirmation message**.**
* **issue\_book(self, book\_title)::** This method is used to issue a book. It checks if the book\_title is in the self.books list. If it is, the book is removed from the list, and a confirmation message is printed. If the book is not found (the edge case of issuing an unavailable book), it prints an error message.
* **display\_books(self)::** This method prints the titles of all the books currently available in the self.books list. If the list is empty, it indicates that the library has no books