ASSIGNMENT-6.1

Name: D. Sravika Reddy Hall-Ticket No: 2403a510d0

Batch No: 05 Course: Ai assisted Coding

Task Description #1 (Classes – Employee Management)

- Task: Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.
- Instructions:

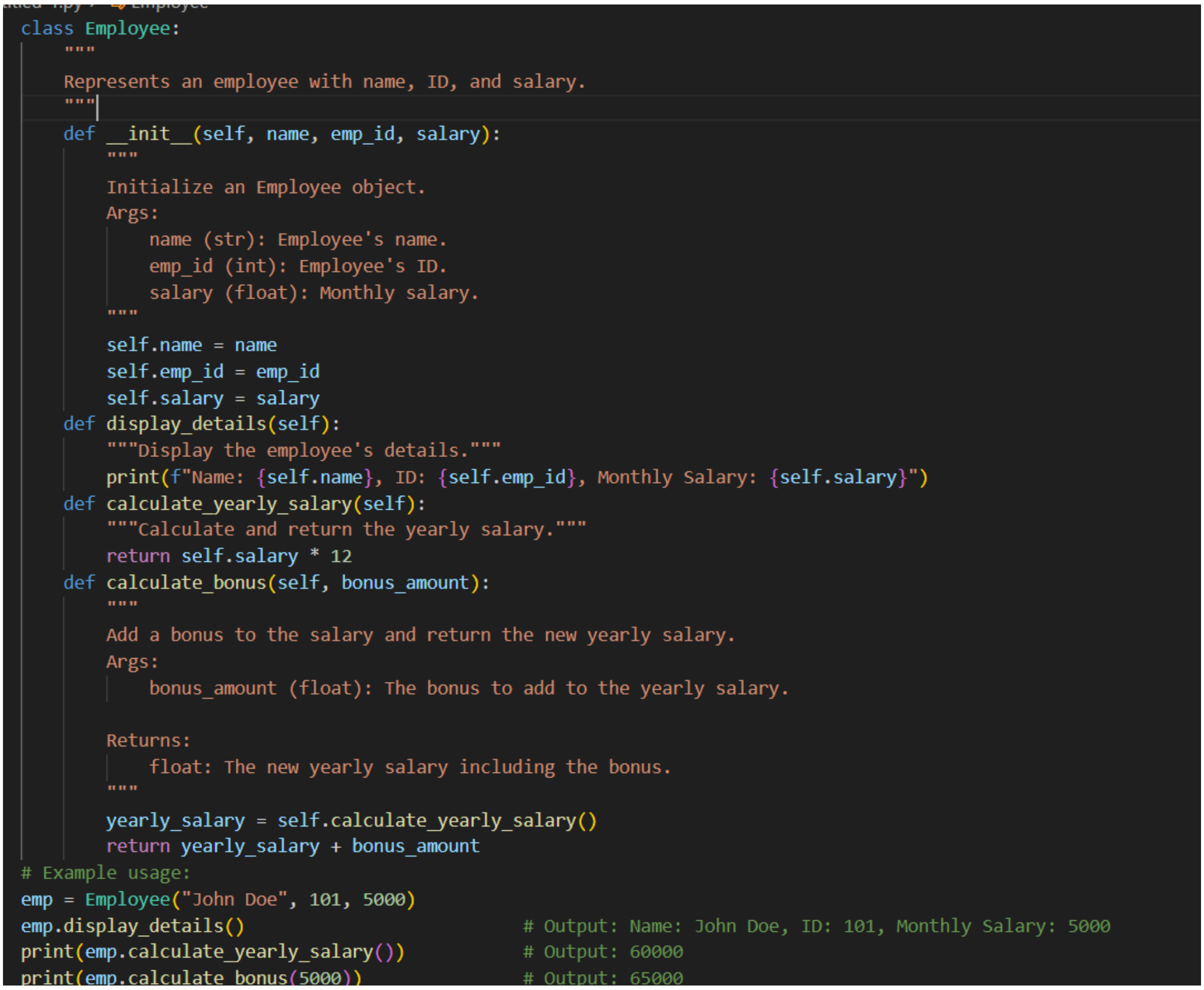
o Prompt AI to generate the Employee class.
o Analyze the generated code for correctness and structure.
o Ask AI to add a method to give a bonus and recalculate salary.

Expected Output #1:

- A class with constructor, display_details(), and calculate_bonus() methods

#Prompt: To create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.

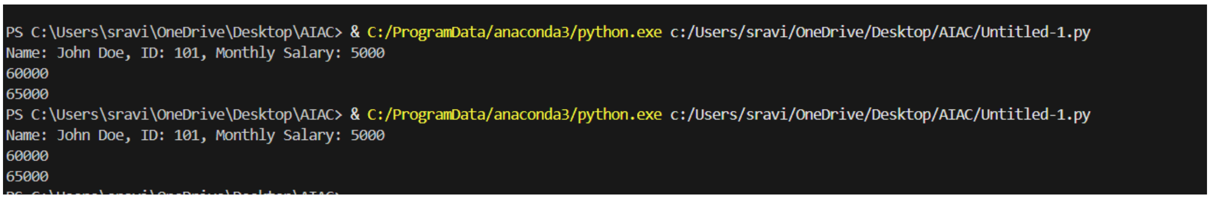
Code:



```python
class Employee:
    """
    Represents an employee with name, ID, and salary.
    """
    def __init__(self, name, emp_id, salary):
        """
        Initialize an Employee object.
        Args:
            name (str): Employee's name.
            emp_id (int): Employee's ID.
            salary (float): Monthly salary.
        """
        self.name = name
        self.emp_id = emp_id
        self.salary = salary
    def display_details(self):
        """Display the employee's details."""
        print(f"Name: {self.name}, ID: {self.emp_id}, Monthly Salary: {self.salary}")
    def calculate_yearly_salary(self):
        """Calculate and return the yearly salary."""
        return self.salary * 12
    def calculate_bonus(self, bonus_amount):
        """
        Add a bonus to the salary and return the new yearly salary.
        Args:
            bonus_amount (float): The bonus to add to the yearly salary.

        Returns:
            float: The new yearly salary including the bonus.
        """
        yearly_salary = self.calculate_yearly_salary()
        return yearly_salary + bonus_amount
# Example usage:
emp = Employee("John Doe", 101, 5000)
emp.display_details()                          # Output: Name: John Doe, ID: 101, Monthly Salary: 5000
print(emp.calculate_yearly_salary())           # Output: 60000
print(emp.calculate_bonus(5000))               # Output: 65000
```

Output:



```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sravi/OneDrive/Desktop/AIAC/Untitled-1.py
Name: John Doe, ID: 101, Monthly Salary: 5000
60000
65000
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sravi/OneDrive/Desktop/AIAC/Untitled-1.py
Name: John Doe, ID: 101, Monthly Salary: 5000
60000
65000
```

Explanation:

- The Employee class models an employee with three attributes: name, emp_id, and salary.
- The __init__ method initializes these attributes when a new Employee object is created.

- The display_details() method prints the employee's name, ID, and monthly salary in a readable format.
- The calculate_yearly_salary() method returns the annual salary by multiplying the monthly salary by 12.
- The calculate_bonus(bonus_amount) method adds a given bonus to the yearly salary and returns the total.
- In the example usage, an Employee object is created for "John Doe" with ID 101 and a monthly salary of 5000. The details are displayed, the yearly salary is printed (60000), and the yearly salary including a 5000 bonus is printed (65000).
- The code is modular, readable, and demonstrates basic object-oriented programming principles.

Task Description #2 (Loops – Automorphic Numbers in a Range)

- Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
- Instructions:

o Get AI-generated code to list Automorphic numbers using a for loop.

o Analyze the correctness and efficiency of the generated logic.

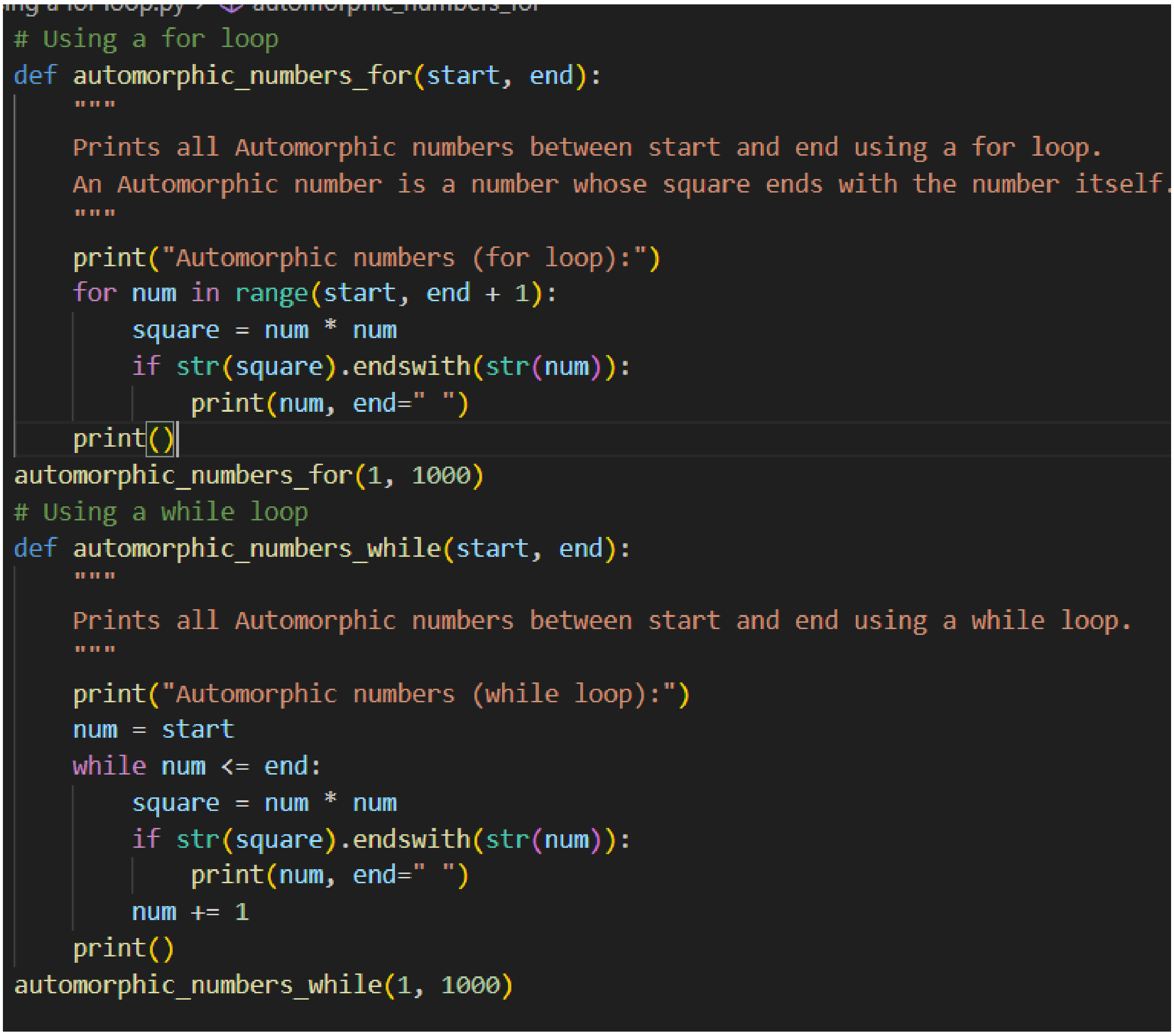
o Ask AI to regenerate using a while loop and compare both implementations.

Expected Output #2:

- Correct implementation that lists Automorphic numbers using both loop types, with explanation.

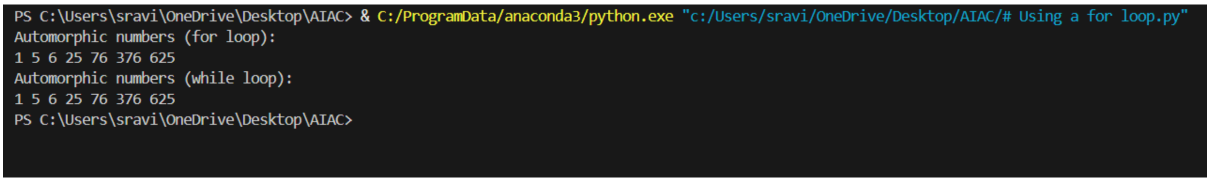
#Prompt: a function that displays all Automorphic numbers between 1 and 1000 using a for loop.

Code:



```python
# Using a for loop
def automorphic_numbers_for(start, end):
    """
    Prints all Automorphic numbers between start and end using a for loop.
    An Automorphic number is a number whose square ends with the number itself.
    """
    print("Automorphic numbers (for loop):")
    for num in range(start, end + 1):
        square = num * num
        if str(square).endswith(str(num)):
            print(num, end=" ")
    print()
automorphic_numbers_for(1, 1000)
# Using a while loop
def automorphic_numbers_while(start, end):
    """
    Prints all Automorphic numbers between start and end using a while loop.
    """
    print("Automorphic numbers (while loop):")
    num = start
    while num <= end:
        square = num * num
        if str(square).endswith(str(num)):
            print(num, end=" ")
        num += 1
    print()
automorphic_numbers_while(1, 1000)
```

Output:



```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe "c:/Users/sravi/OneDrive/Desktop/AIAC/# Using a for loop.py"
Automorphic numbers (for loop):
1 5 6 25 76 376 625
Automorphic numbers (while loop):
1 5 6 25 76 376 625
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

Explanation:

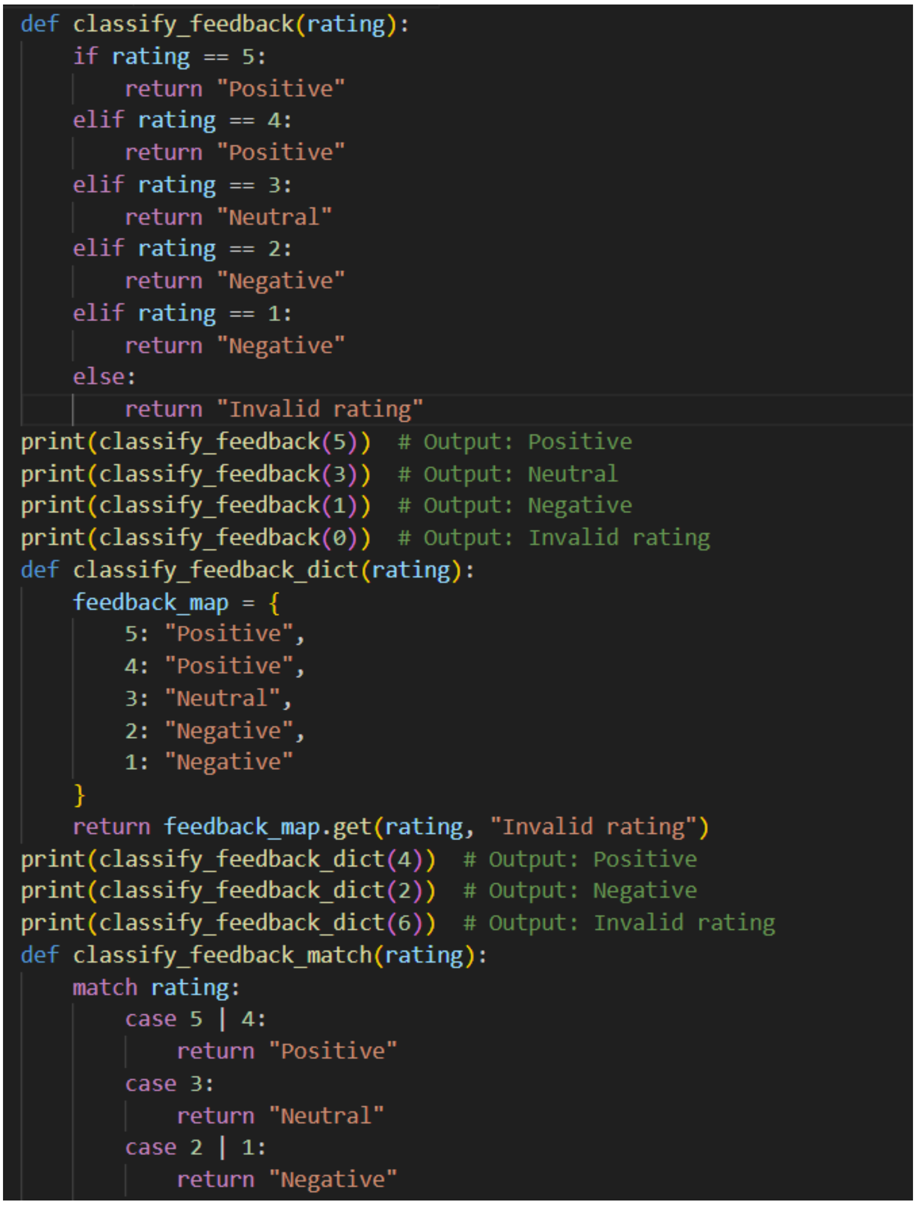
- An **Automorphic number** is a number whose square ends with the number itself (e.g., $76^2 = 5776$, ends with 76).
- Both implementations correctly find and print all Automorphic numbers between 1 and 1000.

- The for loop is concise and Pythonic for a known range.
- The while loop is functionally equivalent but uses explicit incrementing, which is less idiomatic in Python for fixed ranges.
- Both are efficient for this small range, and results are identical.

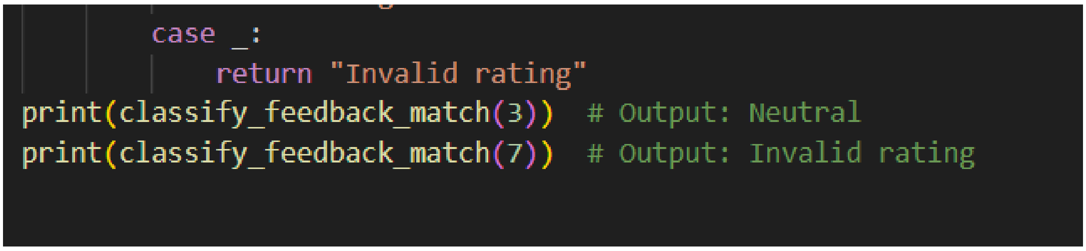
Task Description #3 (Conditional Statements – Online Shopping Feedback Classification)
• Task: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).
• Instructions:
o Generate initial code using nested if-elif-else.
o Analyze correctness and readability.
o Ask AI to rewrite using dictionary-based or match-case structure.
Expected Output #3:
• Feedback classification function with explanation and an alternative approach.

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

Code:

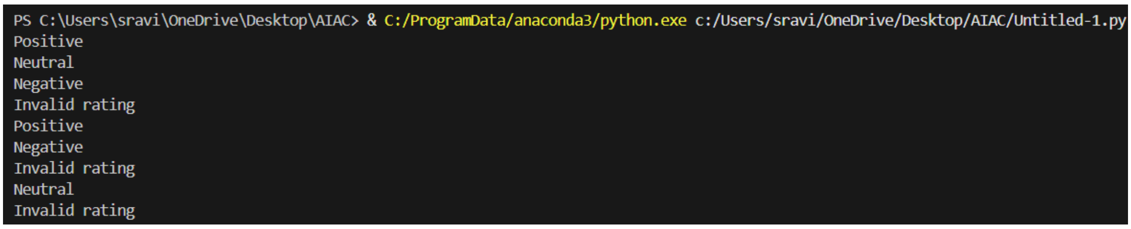


```python
def classify_feedback(rating):
    if rating == 5:
        return "Positive"
    elif rating == 4:
        return "Positive"
    elif rating == 3:
        return "Neutral"
    elif rating == 2:
        return "Negative"
    elif rating == 1:
        return "Negative"
    else:
        return "Invalid rating"
print(classify_feedback(5))  # Output: Positive
print(classify_feedback(3))  # Output: Neutral
print(classify_feedback(1))  # Output: Negative
print(classify_feedback(0))  # Output: Invalid rating
def classify_feedback_dict(rating):
    feedback_map = {
        5: "Positive",
        4: "Positive",
        3: "Neutral",
        2: "Negative",
        1: "Negative"
    }
    return feedback_map.get(rating, "Invalid rating")
print(classify_feedback_dict(4))  # Output: Positive
print(classify_feedback_dict(2))  # Output: Negative
print(classify_feedback_dict(6))  # Output: Invalid rating
def classify_feedback_match(rating):
    match rating:
        case 5 | 4:
            return "Positive"
        case 3:
            return "Neutral"
        case 2 | 1:
            return "Negative"
```



```python
        case _:
            return "Invalid rating"
print(classify_feedback_match(3))  # Output: Neutral
print(classify_feedback_match(7))  # Output: Invalid rating
```

Output:



```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sravi/OneDrive/Desktop/AIAC/Untitled-1.py
Positive
Neutral
Negative
Invalid rating
Positive
Negative
Invalid rating
Neutral
Invalid rating
```

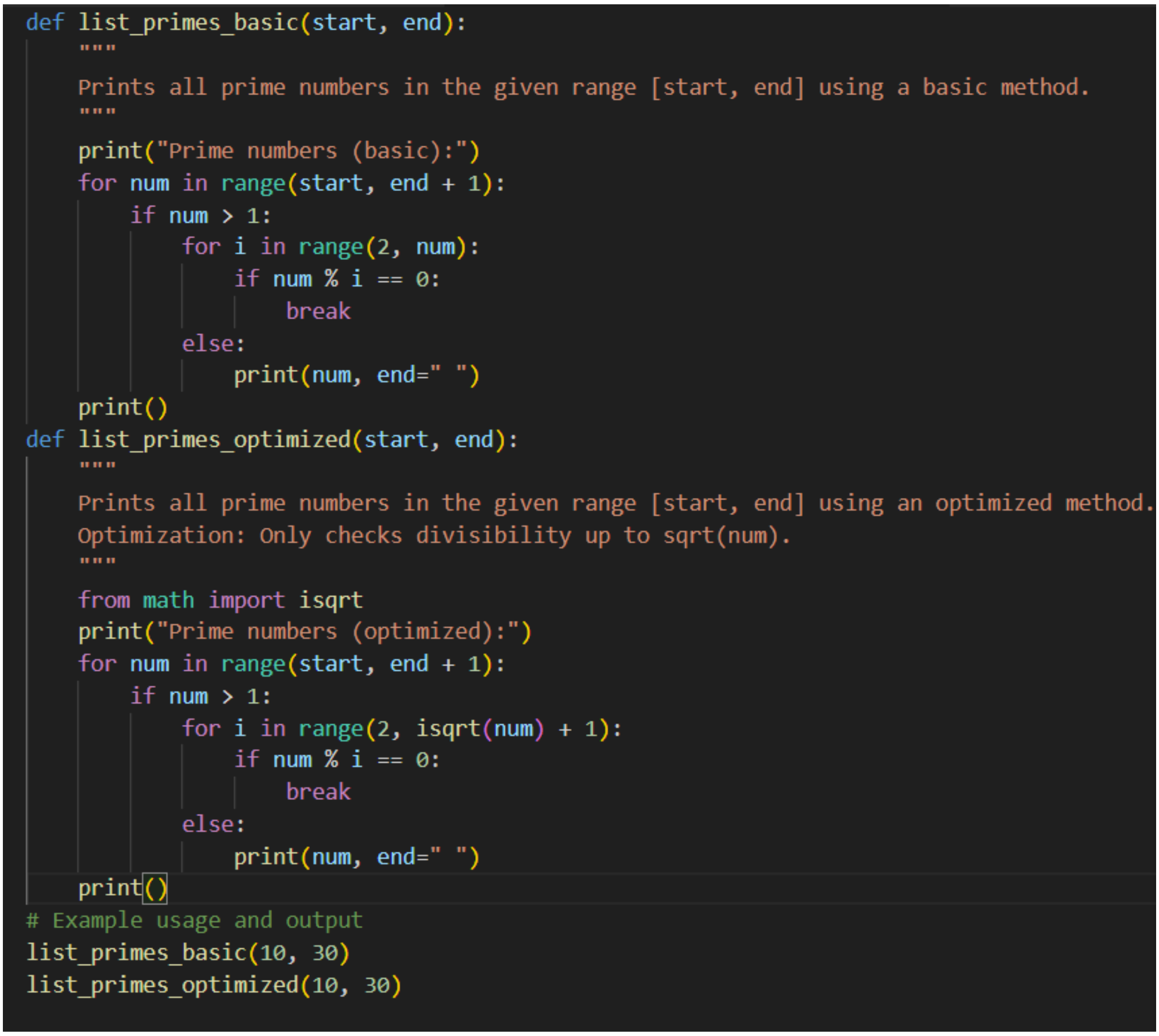
Explanation:

- The nested if-elif-else version is correct but repetitive.
- The dictionary-based approach is more concise and readable.
- The match-case approach (Python 3.10+) is clean and expressive for pattern matching.
- All methods correctly classify ratings and handle invalid input.

Expected Output #4:

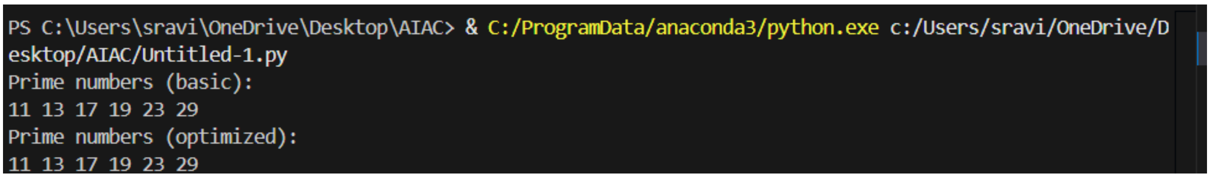
- Python program that lists all prime numbers within a given range, with an optimized version and explanation

#Prompt: Lists all prime numbers within a given range.



```python
def list_primes_basic(start, end):
    """
    Prints all prime numbers in the given range [start, end] using a basic method.
    """
    print("Prime numbers (basic):")
    for num in range(start, end + 1):
        if num > 1:
            for i in range(2, num):
                if num % i == 0:
                    break
            else:
                print(num, end=" ")
    print()
def list_primes_optimized(start, end):
    """
    Prints all prime numbers in the given range [start, end] using an optimized method.
    Optimization: Only checks divisibility up to sqrt(num).
    """
    from math import isqrt
    print("Prime numbers (optimized):")
    for num in range(start, end + 1):
        if num > 1:
            for i in range(2, isqrt(num) + 1):
                if num % i == 0:
                    break
            else:
                print(num, end=" ")
    print()
# Example usage and output
list_primes_basic(10, 30)
list_primes_optimized(10, 30)
```

Output:



```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sravi/OneDrive/D
esktop/AIAC/Untitled-1.py
Prime numbers (basic):
11 13 17 19 23 29
Prime numbers (optimized):
11 13 17 19 23 29
```

Explanation:

- The list_primes_basic function checks each number in the range for primality by testing divisibility from 2 up to num-1.
- The list_primes_optimized function improves efficiency by only checking divisibility up to the integer square root of each number, reducing unnecessary checks.
- Both functions correctly list all prime numbers in the given range, but the optimized version is faster for larger ranges.

Task Description #5 (Classes – Library System)

- Task: Use AI to build a Library class with methods to add_book(), issue_book(), and display_books().
- Instructions:

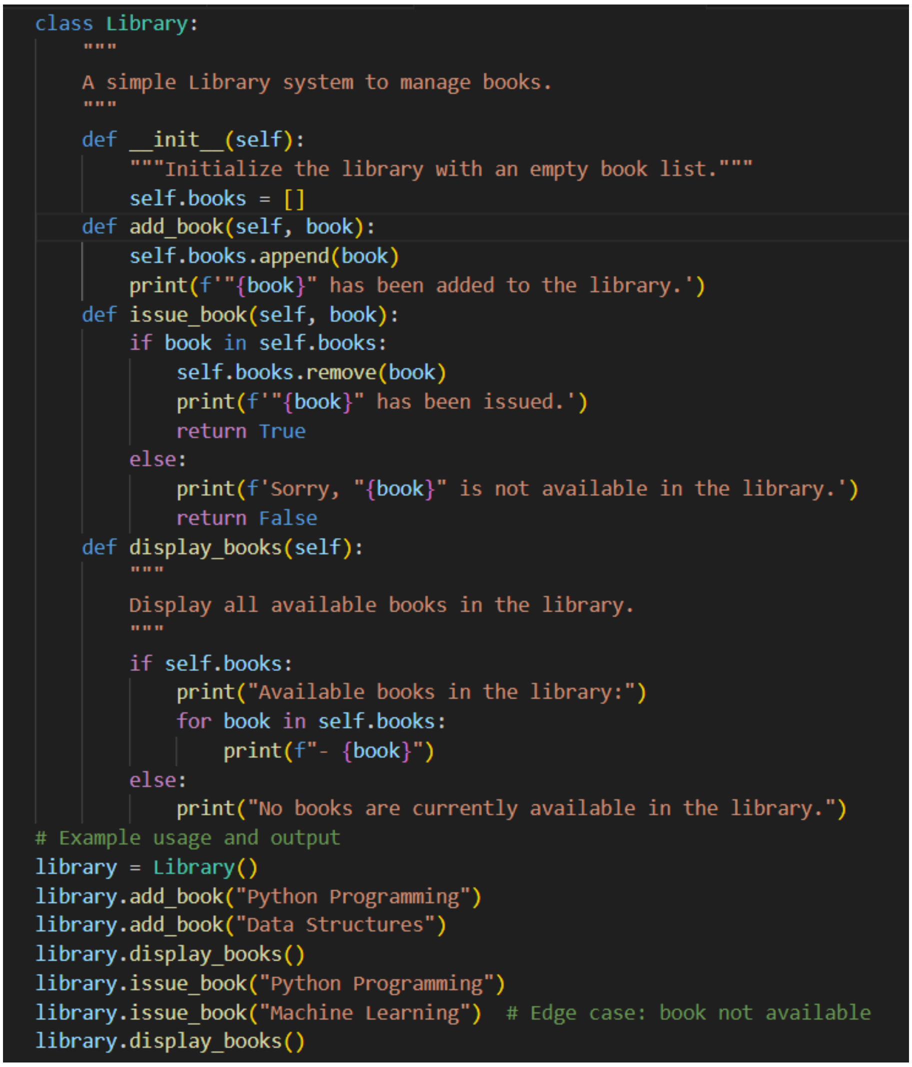
o Generate Library class code using AI.
o Analyze if methods handle edge cases (e.g., issuing unavailable books).
o Ask AI to add comments and documentation.

Expected Output #5:

- Library class with all methods, inline comments, and explanation

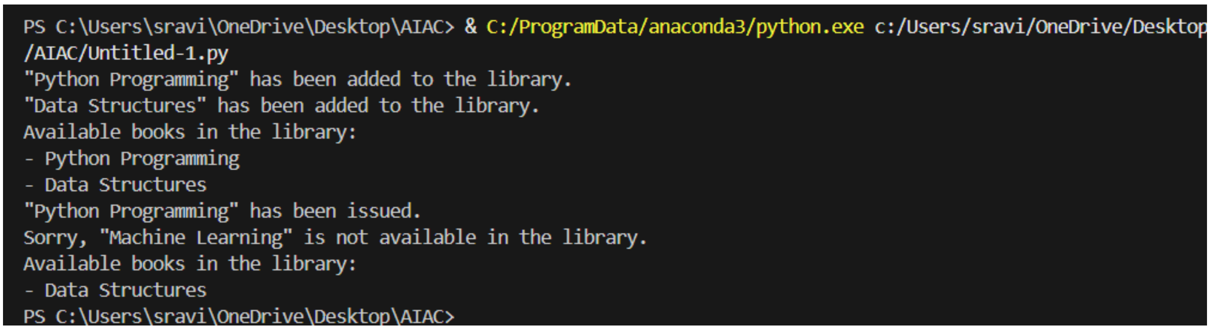
#Prompt: Build a Library class with methods to add_book(), issue_book(), and display_books().

Code:



```python
class Library:
    """
    A simple Library system to manage books.
    """
    def __init__(self):
        """Initialize the library with an empty book list."""
        self.books = []
    def add_book(self, book):
        self.books.append(book)
        print(f'"{book}" has been added to the library.')
    def issue_book(self, book):
        if book in self.books:
            self.books.remove(book)
            print(f'"{book}" has been issued.')
            return True
        else:
            print(f'Sorry, "{book}" is not available in the library.')
            return False
    def display_books(self):
        """
        Display all available books in the library.
        """
        if self.books:
            print("Available books in the library:")
            for book in self.books:
                print(f"- {book}")
        else:
            print("No books are currently available in the library.")
# Example usage and output
library = Library()
library.add_book("Python Programming")
library.add_book("Data Structures")
library.display_books()
library.issue_book("Python Programming")
library.issue_book("Machine Learning")  # Edge case: book not available
library.display_books()
```

Output:



```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sravi/OneDrive/Desktop
/AIAC/Untitled-1.py
"Python Programming" has been added to the library.
"Data Structures" has been added to the library.
Available books in the library:
- Python Programming
- Data Structures
"Python Programming" has been issued.
Sorry, "Machine Learning" is not available in the library.
Available books in the library:
- Data Structures
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

Explanation:

- The Library class manages a list of books.
- add_book() adds a book and confirms addition.

- issue_book() checks if the book is available before issuing; handles the edge case where the book is not present.
- display_books() lists all available books or notifies if none are available.
- Inline comments and docstrings explain each method and its parameters