AI ASSISTED CODING

ASSIGNMENT-6.1

2403A51260

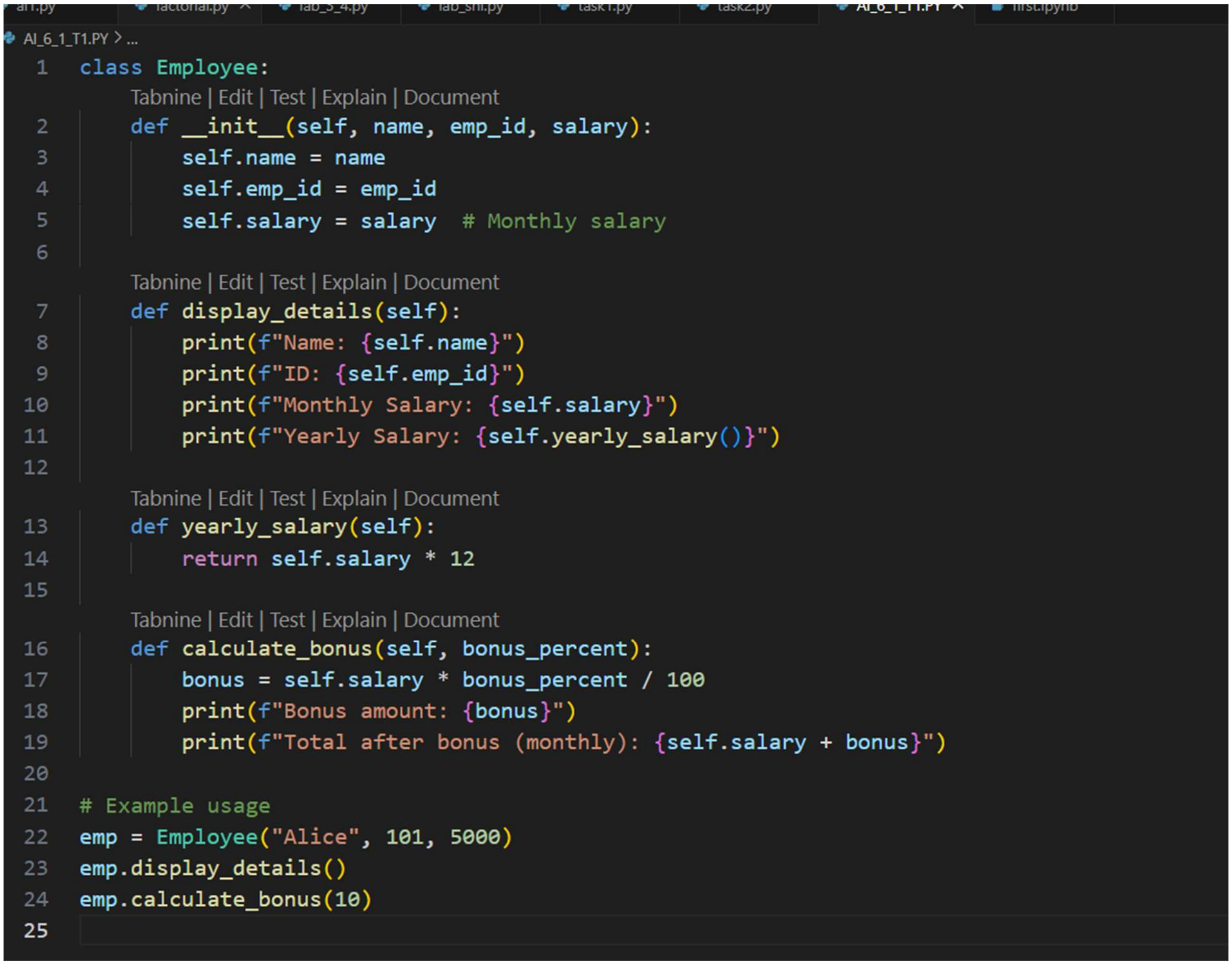
SWAPNA MADISHETTI

BATCH-11

Task-1:

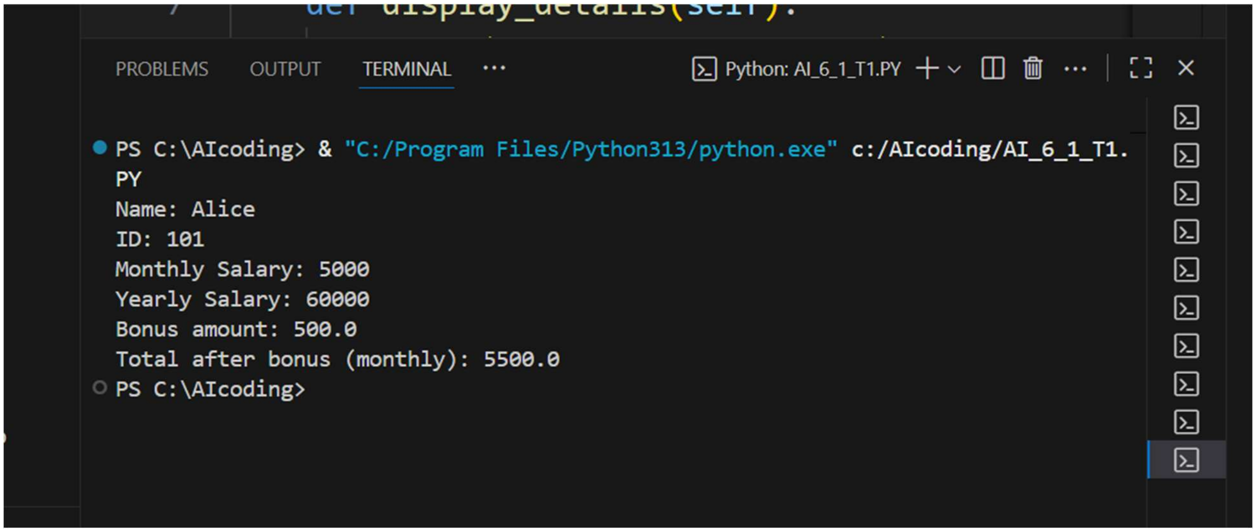
Prompt: Create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary. class should contain constructor, display_details(), and calculate_bonus() methods.

Code:



```python
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name
        self.emp_id = emp_id
        self.salary = salary  # Monthly salary

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"ID: {self.emp_id}")
        print(f"Monthly Salary: {self.salary}")
        print(f"Yearly Salary: {self.yearly_salary()}")

    def yearly_salary(self):
        return self.salary * 12

    def calculate_bonus(self, bonus_percent):
        bonus = self.salary * bonus_percent / 100
        print(f"Bonus amount: {bonus}")
        print(f"Total after bonus (monthly): {self.salary + bonus}")

# Example usage
emp = Employee("Alice", 101, 5000)
emp.display_details()
emp.calculate_bonus(10)

```

Output:

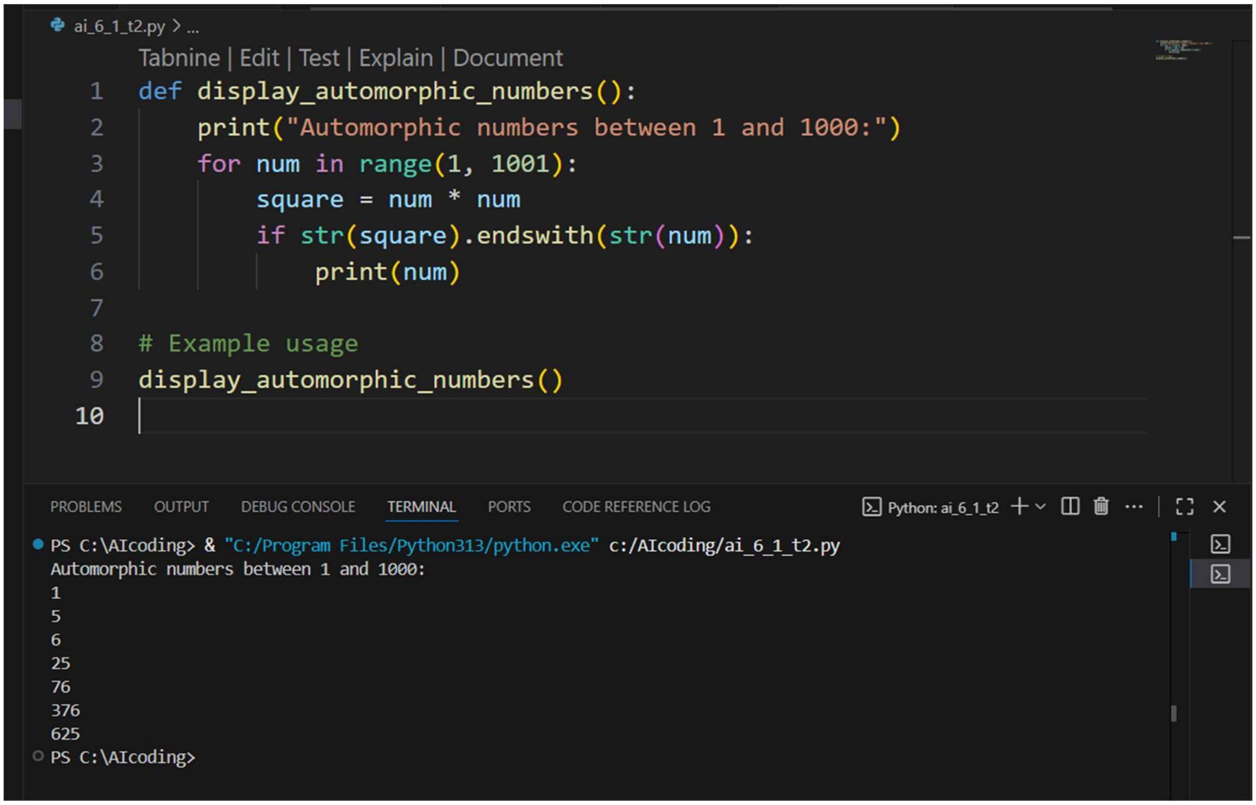


```
PS C:\AIcoding> & "C:/Program Files/Python313/python.exe" c:/AIcoding/AI_6_1_T1.
PY
Name: Alice
ID: 101
Monthly Salary: 5000
Yearly Salary: 60000
Bonus amount: 500.0
Total after bonus (monthly): 5500.0
PS C:\AIcoding>
```

Task-2:

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

Code & Output:

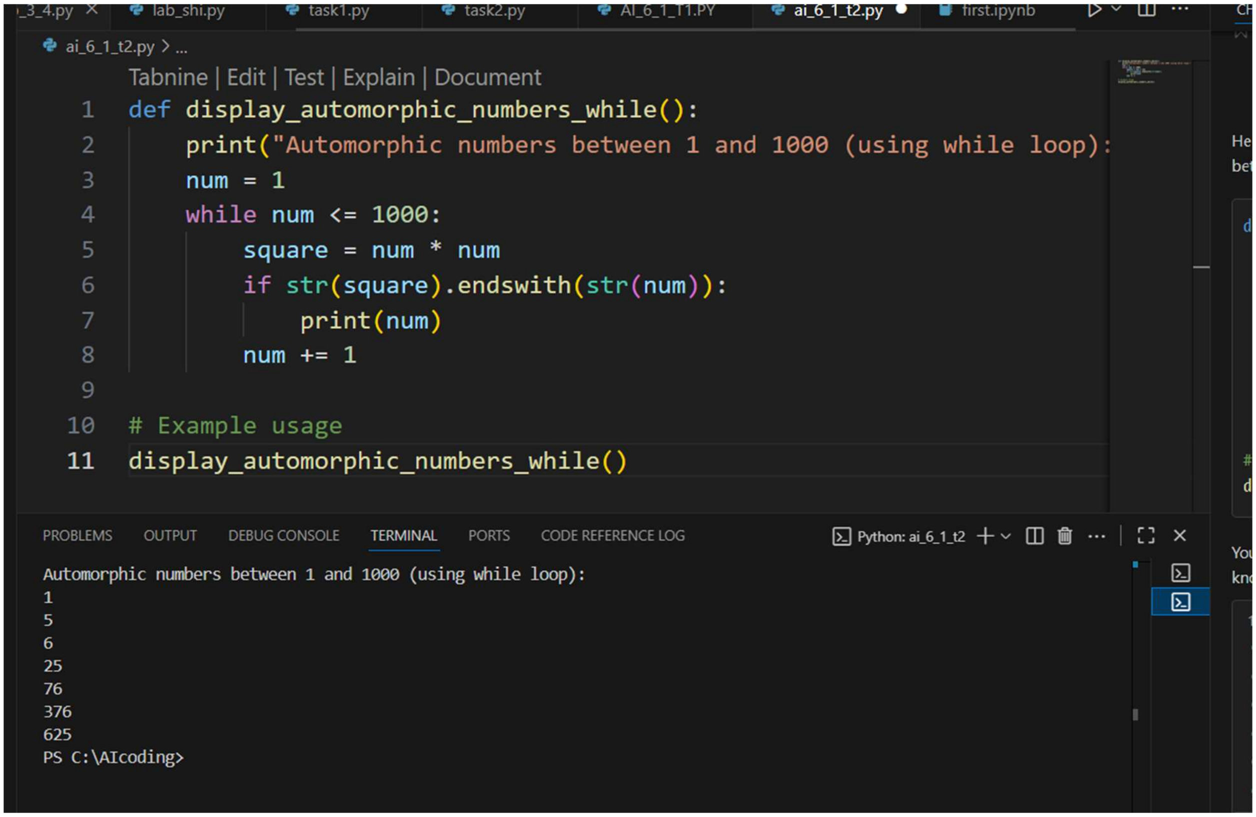


```python
def display_automorphic_numbers():
    print("Automorphic numbers between 1 and 1000:")
    for num in range(1, 1001):
        square = num * num
        if str(square).endswith(str(num)):
            print(num)

# Example usage
display_automorphic_numbers()
```

```
PS C:\AIcoding> & "C:/Program Files/Python313/python.exe" c:/AIcoding/ai_6_1_t2.py
Automorphic numbers between 1 and 1000:
1
5
6
25
76
376
625
PS C:\AIcoding>
```

Prompt-2: Now generate the code using while loop.

Code & Output:



```python
def display_automorphic_numbers_while():
    print("Automorphic numbers between 1 and 1000 (using while loop):
    num = 1
    while num <= 1000:
        square = num * num
        if str(square).endswith(str(num)):
            print(num)
        num += 1

# Example usage
display_automorphic_numbers_while()
```

```
Automorphic numbers between 1 and 1000 (using while loop):
1
5
6
25
76
376
625
PS C:\AIcoding>
```

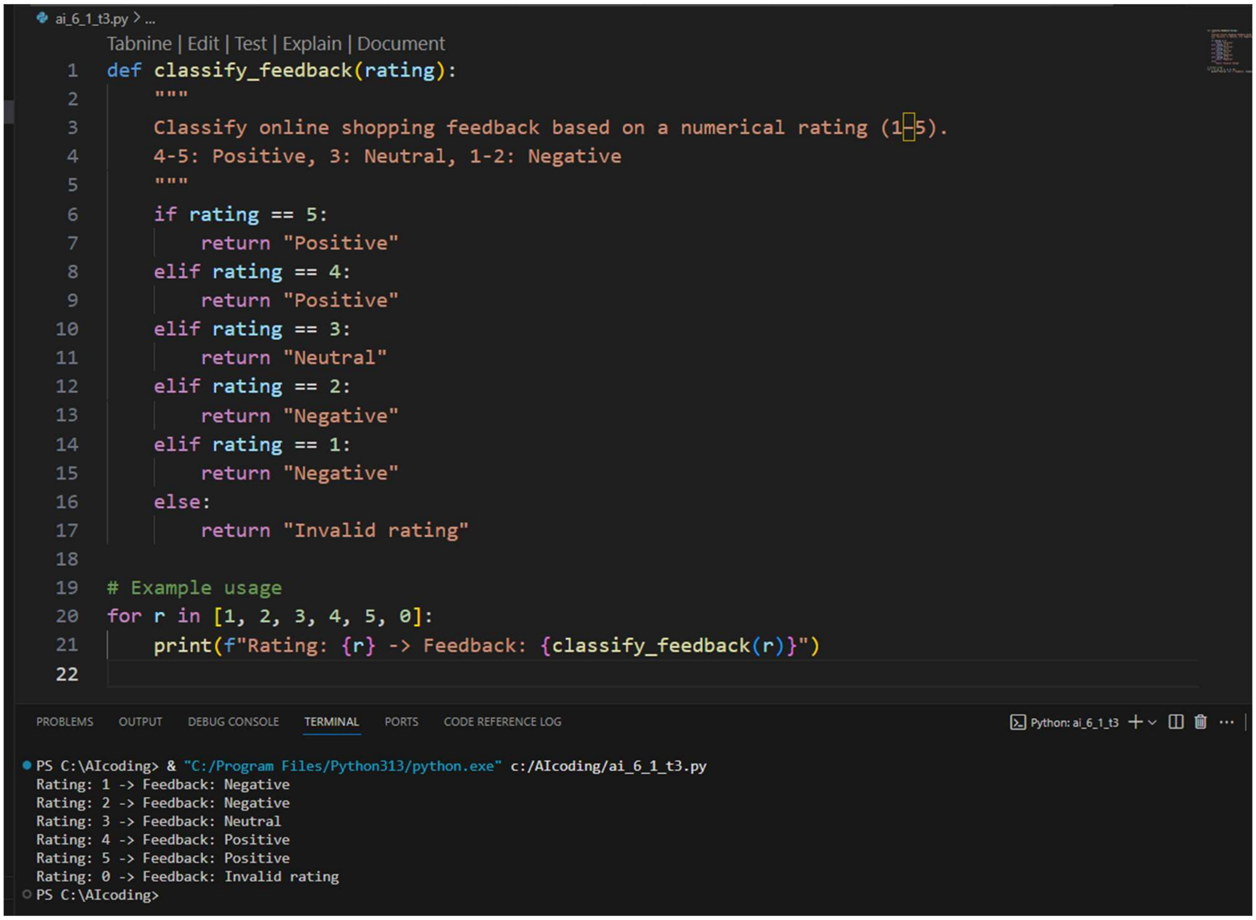
Comparision :

- Both codes achieve the same result: listing Automorphic numbers from 1 to 1000.
- The for loop is more concise and idiomatic for a known range.
- The while loop gives more control over the loop variable and is useful when the end condition is more complex or not based on a simple range.

Task-3:

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 & Output:

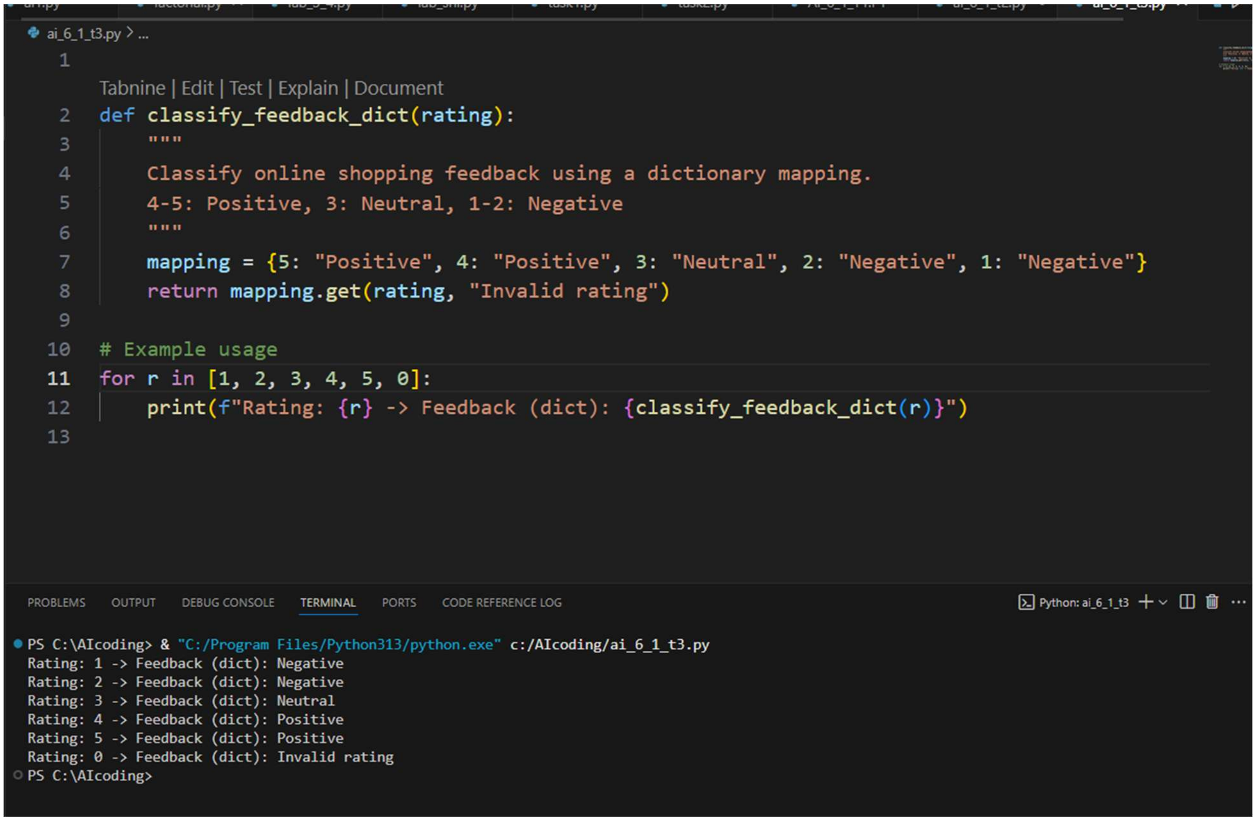


```python
def classify_feedback(rating):
    """
    Classify online shopping feedback based on a numerical rating (1-5).
    4-5: Positive, 3: Neutral, 1-2: Negative
    """
    if rating == 5:
        return "Positive"
    elif rating == 4:
        return "Positive"
    elif rating == 3:
        return "Neutral"
    elif rating == 2:
        return "Negative"
    elif rating == 1:
        return "Negative"
    else:
        return "Invalid rating"

# Example usage
for r in [1, 2, 3, 4, 5, 0]:
    print(f"Rating: {r} -> Feedback: {classify_feedback(r)}")

```

```
PS C:\AIcoding> & "C:/Program Files/Python313/python.exe" c:/AIcoding/ai_6_1_t3.py
Rating: 1 -> Feedback: Negative
Rating: 2 -> Feedback: Negative
Rating: 3 -> Feedback: Neutral
Rating: 4 -> Feedback: Positive
Rating: 5 -> Feedback: Positive
Rating: 0 -> Feedback: Invalid rating
PS C:\AIcoding>
```

Prompt: Rewrite using dictionary-based structure.

Code & Output:



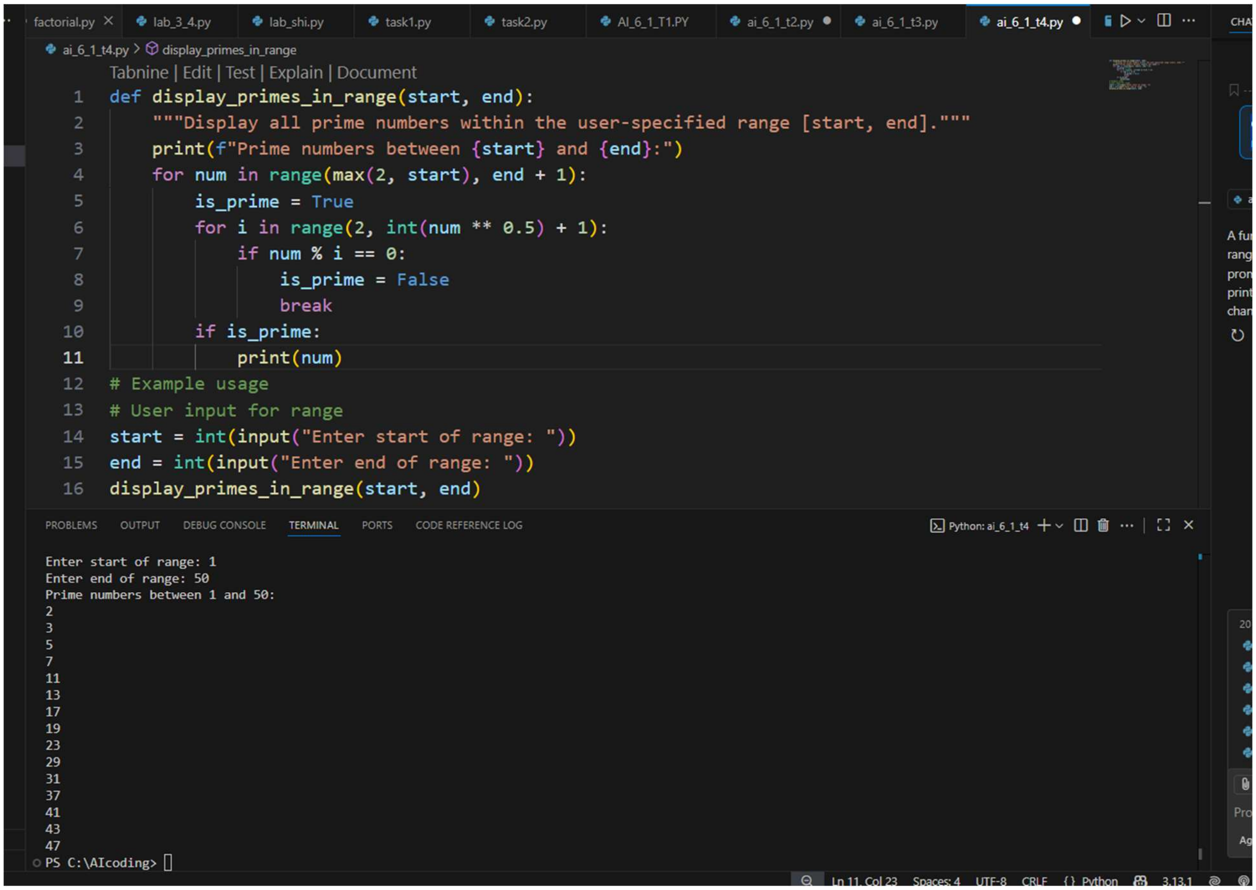
```python

def classify_feedback_dict(rating):
    """
    Classify online shopping feedback using a dictionary mapping.
    4-5: Positive, 3: Neutral, 1-2: Negative
    """
    mapping = {5: "Positive", 4: "Positive", 3: "Neutral", 2: "Negative", 1: "Negative"}
    return mapping.get(rating, "Invalid rating")

# Example usage
for r in [1, 2, 3, 4, 5, 0]:
    print(f"Rating: {r} -> Feedback (dict): {classify_feedback_dict(r)}")

```

```
PS C:\AIcoding> & "C:/Program Files/Python313/python.exe" c:/AIcoding/ai_6_1_t3.py
Rating: 1 -> Feedback (dict): Negative
Rating: 2 -> Feedback (dict): Negative
Rating: 3 -> Feedback (dict): Neutral
Rating: 4 -> Feedback (dict): Positive
Rating: 5 -> Feedback (dict): Positive
Rating: 0 -> Feedback (dict): Invalid rating
PS C:\AIcoding>
```

Task-4:

Prompt: Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).

Code & Output:

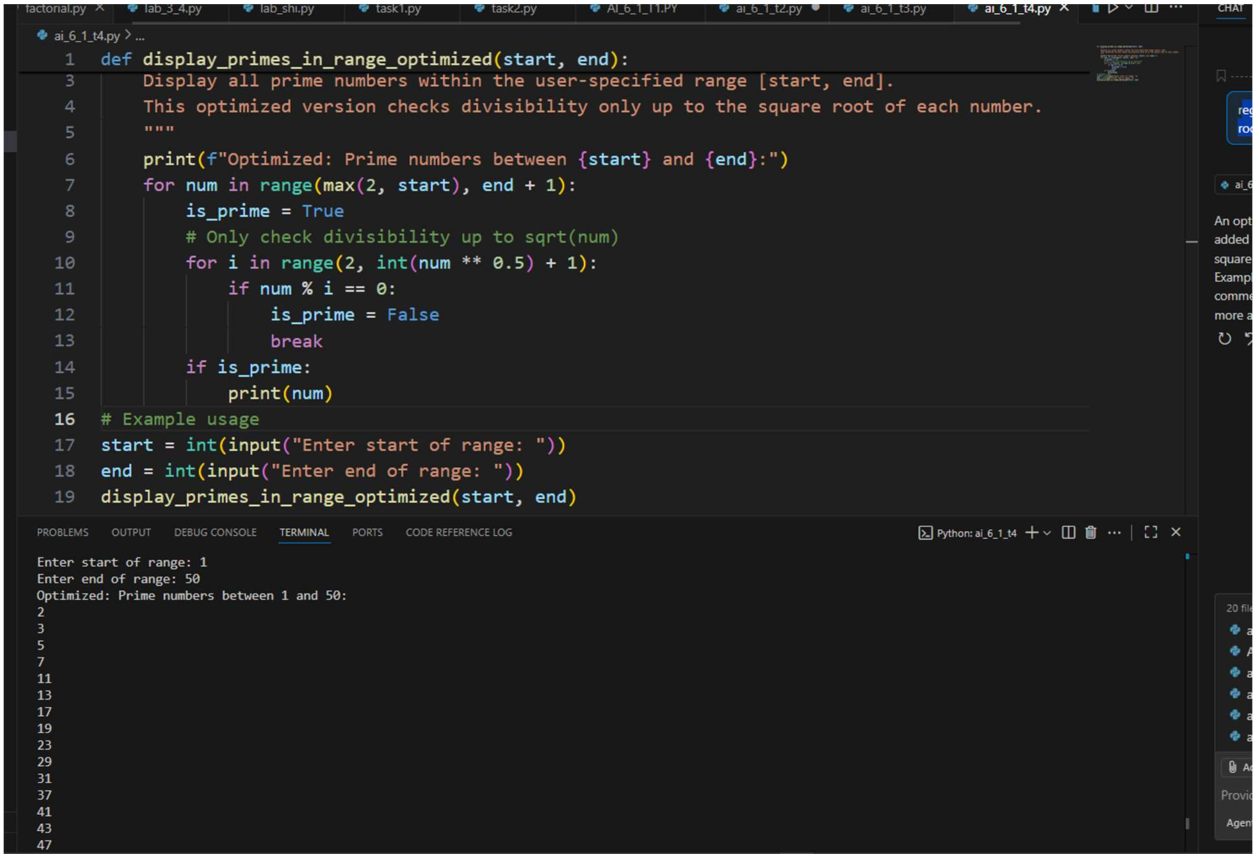


```python
def display_primes_in_range(start, end):
    """Display all prime numbers within the user-specified range [start, end]."""
    print(f"Prime numbers between {start} and {end}:")
    for num in range(max(2, start), end + 1):
        is_prime = True
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            print(num)
# Example usage
# User input for range
start = int(input("Enter start of range: "))
end = int(input("Enter end of range: "))
display_primes_in_range(start, end)
```

```
Enter start of range: 1
Enter end of range: 50
Prime numbers between 1 and 50:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
PS C:\AIcoding> 
```

Prompt-2: Generate an optimized version (e.g., using the square root method). with explanation.

Code & Output:



```python
def display_primes_in_range_optimized(start, end):
    Display all prime numbers within the user-specified range [start, end].
    This optimized version checks divisibility only up to the square root of each number.
    """
    print(f"Optimized: Prime numbers between {start} and {end}:")
    for num in range(max(2, start), end + 1):
        is_prime = True
        # Only check divisibility up to sqrt(num)
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            print(num)
# Example usage
start = int(input("Enter start of range: "))
end = int(input("Enter end of range: "))
display_primes_in_range_optimized(start, end)
```

```
Enter start of range: 1
Enter end of range: 50
Optimized: Prime numbers between 1 and 50:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
```

Explanation:

- The function prints all prime numbers between start and end.
- For each number in the range, it checks if the number is prime.
- Instead of checking all numbers up to num-1, it only checks divisibility up to the square root of num (int(num ** 0.5) + 1). This is because if a number has a factor larger than its square root, the corresponding smaller factor would have already been found.
- If no divisors are found, the number is prime and printed.

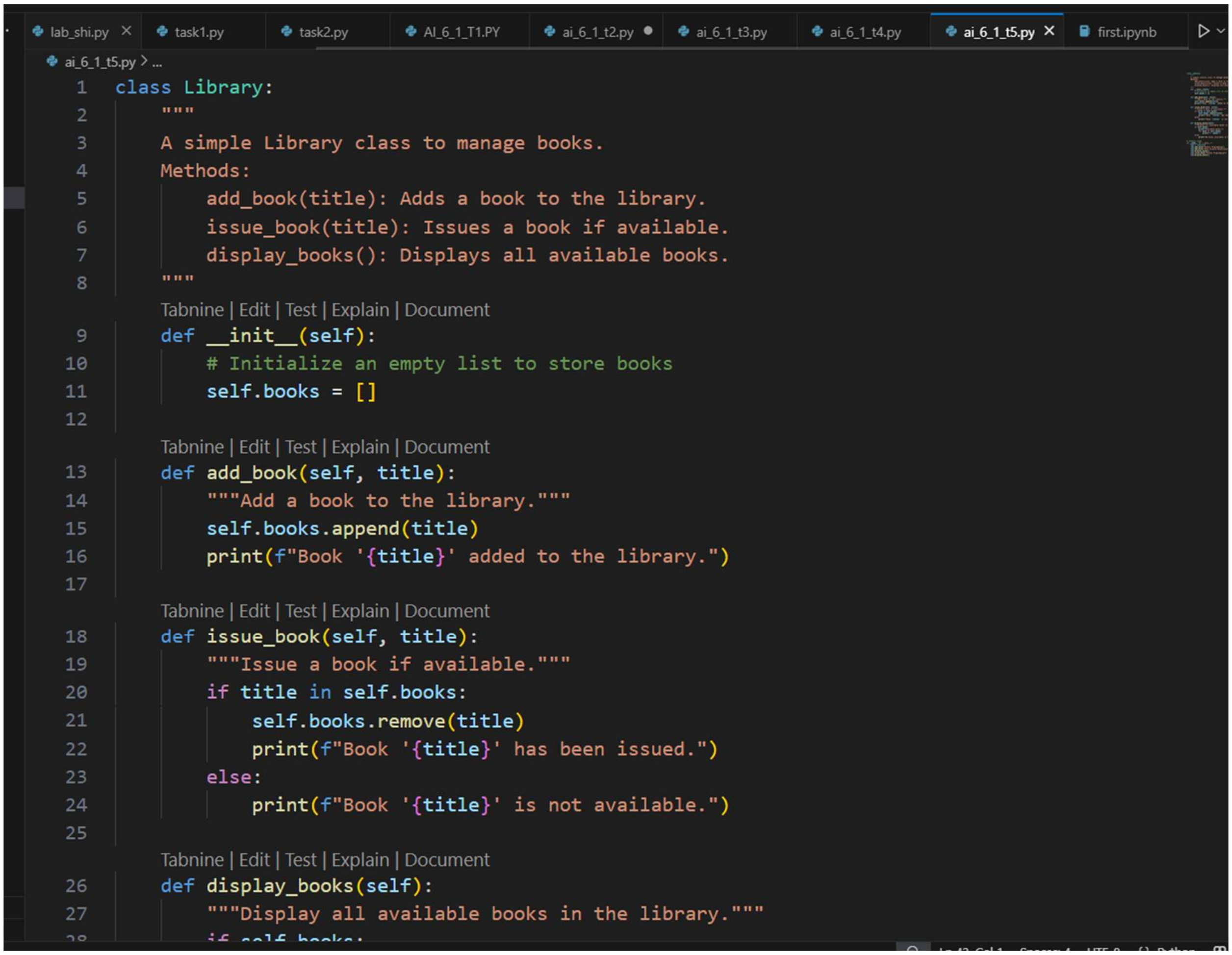
Why is this optimized?

- Checking up to the square root reduces the number of checks, making the function much faster for large ranges compared to checking all possible divisors.

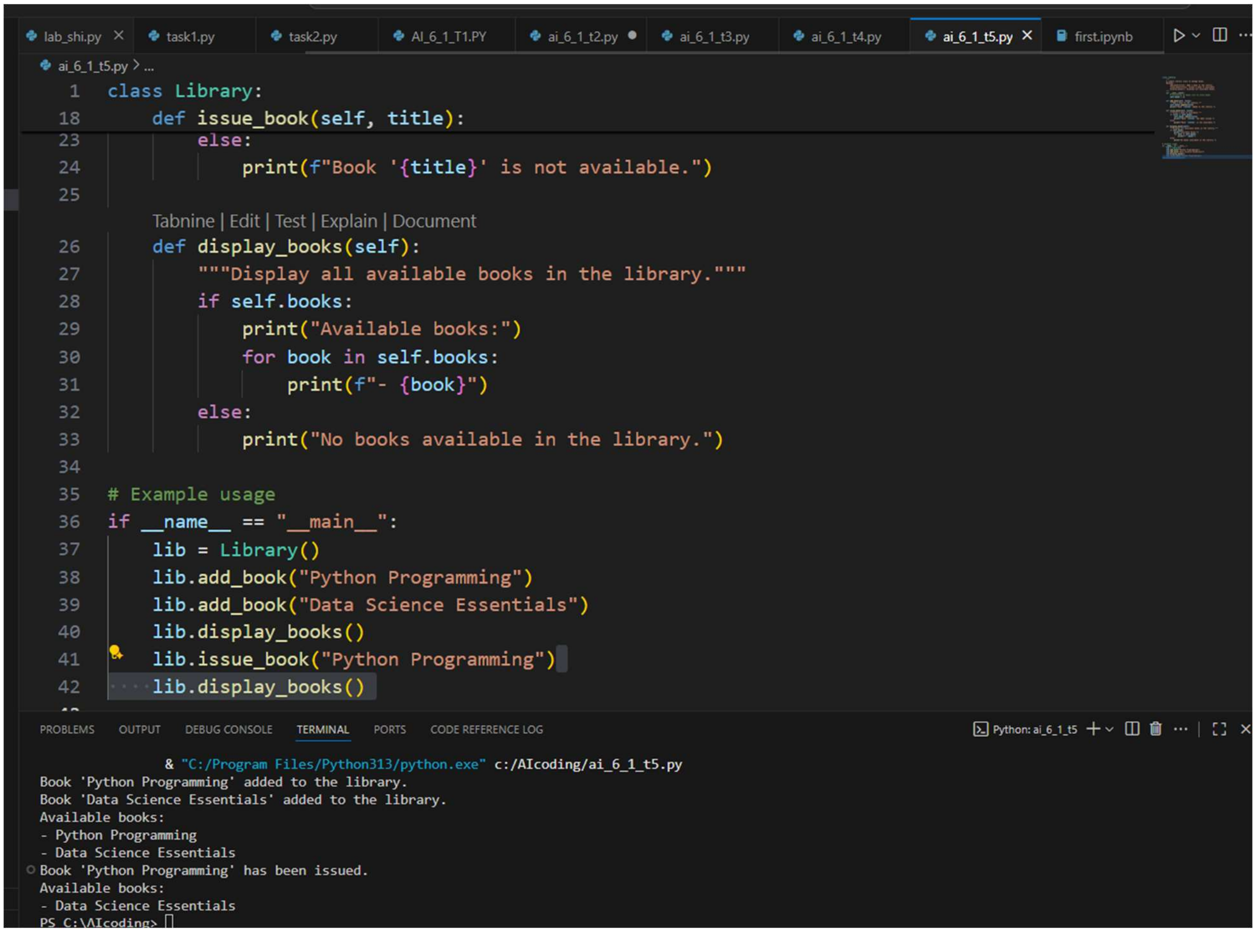
Task-5:

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

Code & Output:



```python
class Library:
    """
    A simple Library class to manage books.
    Methods:
        add_book(title): Adds a book to the library.
        issue_book(title): Issues a book if available.
        display_books(): Displays all available books.
    """
    def __init__(self):
        # Initialize an empty list to store books
        self.books = []

    def add_book(self, title):
        """Add a book to the library."""
        self.books.append(title)
        print(f"Book '{title}' added to the library.")

    def issue_book(self, title):
        """Issue a book if available."""
        if title in self.books:
            self.books.remove(title)
            print(f"Book '{title}' has been issued.")
        else:
            print(f"Book '{title}' is not available.")

    def display_books(self):
        """Display all available books in the library."""
        if self.books:
```



```
class Library:
    def issue_book(self, title):
        else:
            print(f"Book '{title}' is not available.")

    def display_books(self):
        """Display all available books in the library."""
        if self.books:
            print("Available books:")
            for book in self.books:
                print(f"- {book}")
        else:
            print("No books available in the library.")

# Example usage
if __name__ == "__main__":
    lib = Library()
    lib.add_book("Python Programming")
    lib.add_book("Data Science Essentials")
    lib.display_books()
    lib.issue_book("Python Programming")
    lib.display_books()
```

```
                & "C:/Program Files/Python313/python.exe" c:/AIcoding/ai_6_1_t5.py
Book 'Python Programming' added to the library.
Book 'Data Science Essentials' added to the library.
Available books:
- Python Programming
- Data Science Essentials
Book 'Python Programming' has been issued.
Available books:
- Data Science Essentials
PS C:\AIcoding>
```