

LAB – 6 ASSIGNMENT

NAME : SHAIK SALIHA

ROLL NO : 2503A52L13

BATCH : 16

Task -1: 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.

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

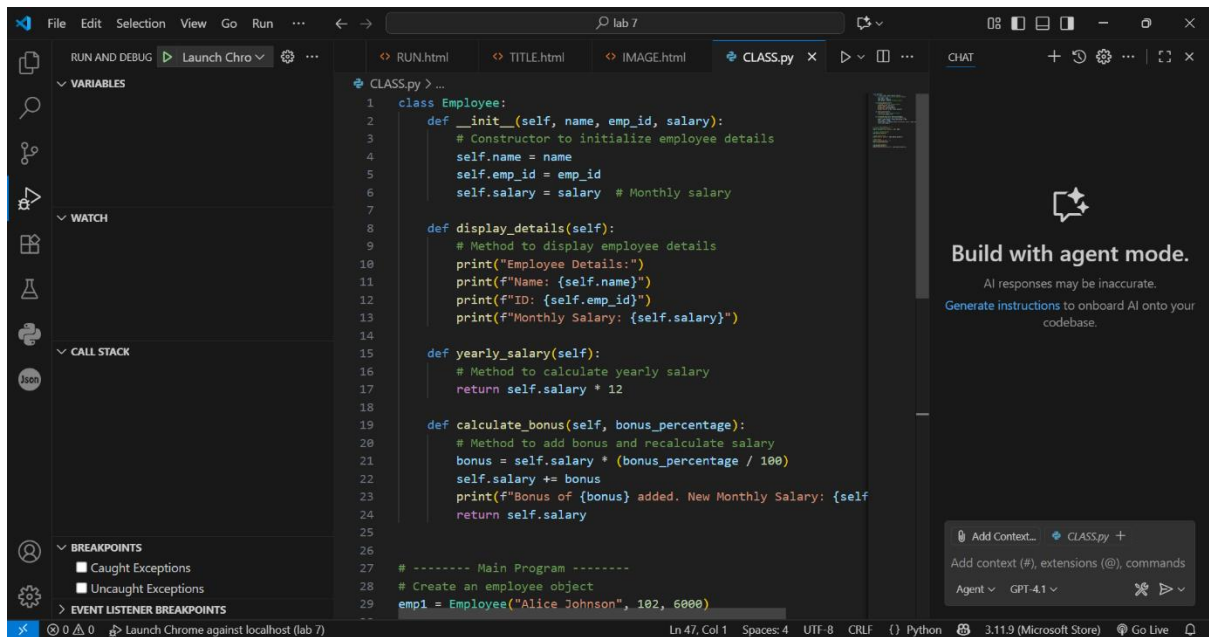
- Instructions:

- 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.

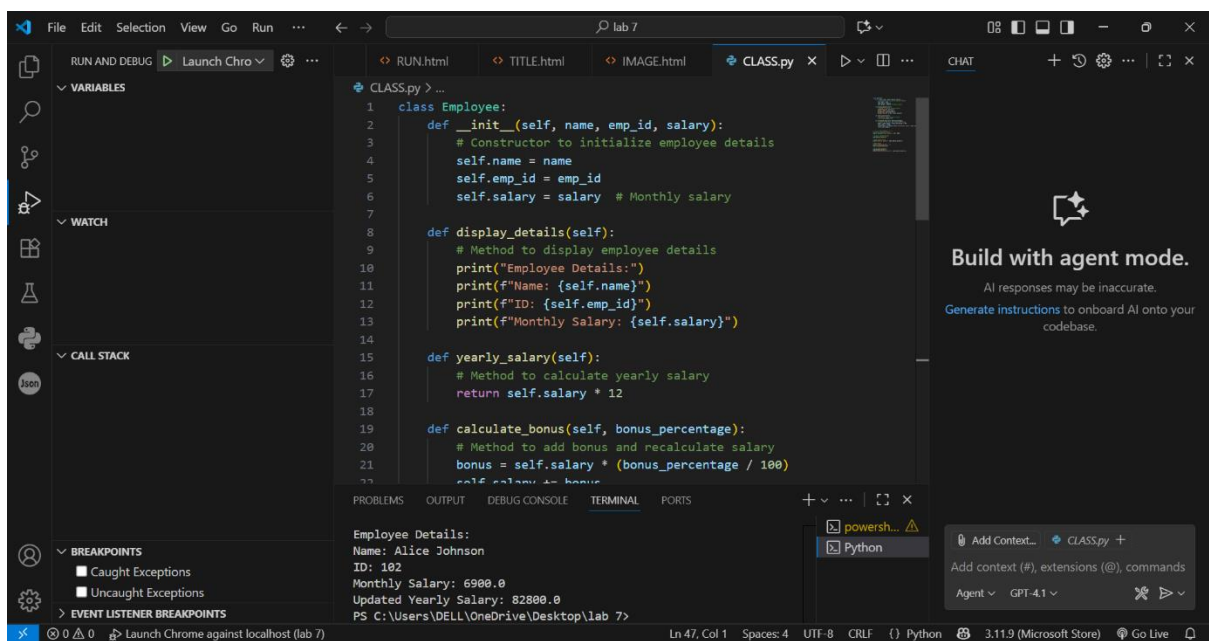
CODE : class Employee:

```
def __init__(self, name, emp_id, salary):  
    # Constructor to initialize employee details  
  
    self.name = name  
  
    self.emp_id = emp_id  
  
    self.salary = salary # Monthly salary  
  
  
def display_details(self):  
    # Method to display employee details  
  
    print("Employee Details:")  
  
    print(f"Name: {self.name}")  
  
    print(f"ID: {self.emp_id}")  
  
    print(f"Monthly Salary: {self.salary}")
```

```
def yearly_salary(self):  
    # Method to calculate yearly salary  
    return self.salary * 12  
  
def calculate_bonus(self, bonus_percentage):  
    # Method to add bonus and recalculate salary  
    bonus = self.salary * (bonus_percentage / 100)  
    self.salary += bonus  
    print(f"Bonus of {bonus} added. New Monthly Salary: {self.salary}")  
    return self.salary  
  
# ----- Main Program -----  
# Create an employee object  
emp1 = Employee("Alice Johnson", 102, 6000)  
  
# Display employee details  
emp1.display_details()  
  
# Show yearly salary  
print("\nYearly Salary:", emp1.yearly_salary())  
  
# Apply bonus  
print("\nApplying Bonus...")  
emp1.calculate_bonus(15)  
  
# Show updated details  
emp1.display_details()  
print("Updated Yearly Salary:", emp1.yearly_salary())
```



OUTPUT :



- **OBSERVATION : Constructor (__init__):** Correctly initializes name, id, and salary.
- **display_details():** Displays employee info in a readable format.
- **yearly_salary():** Multiplies monthly salary by 12 for total yearly salary.

- **calculate_bonus()**: Accepts a bonus percentage, adds it to salary, and updates salary accordingly.

Task -2: 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

PROMPT : 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

CODE : # Function to check if a number is automorphic

```
def is_automorphic(num):
```

```
    square = num ** 2
```

```
    return str(square).endswith(str(num))
```

Using for loop

```
print("Automorphic numbers between 1 and 1000 (for loop):")
```

```
for num in range(1, 1001):
```

```
    if is_automorphic(num):
```

```
        print(num, end=" ")
```

```
print("\n")
```

Using while loop

```
print("Automorphic numbers between 1 and 1000 (while loop):")
```

```
num = 1
```

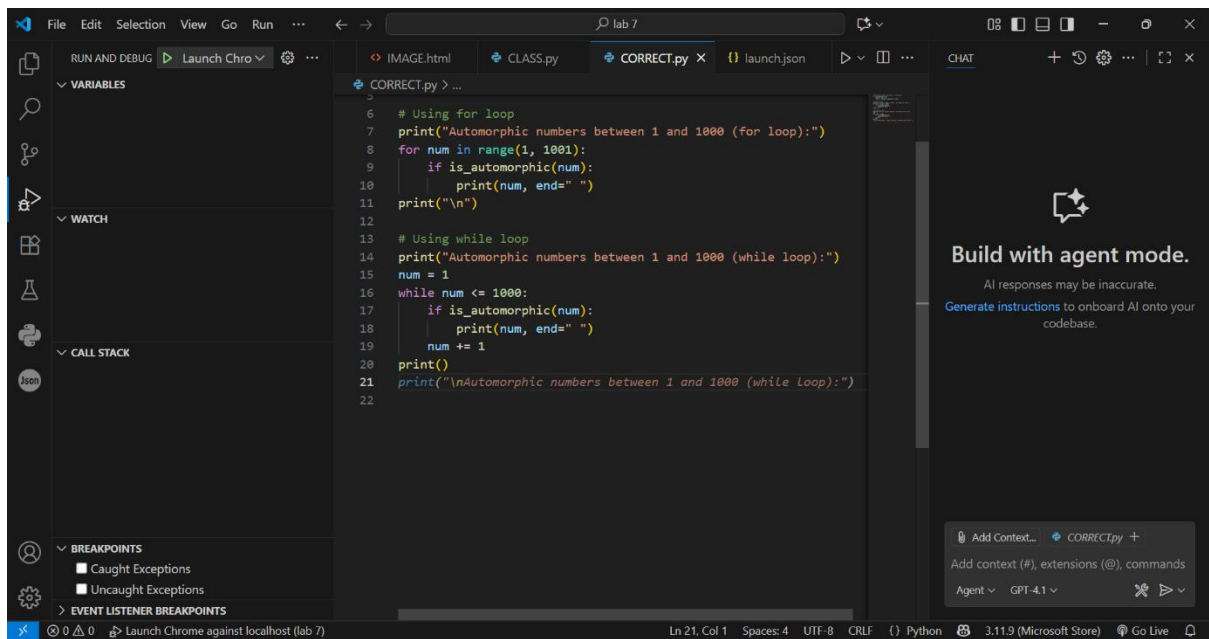
```
while num <= 1000:
```

```
    if is_automorphic(num):
```

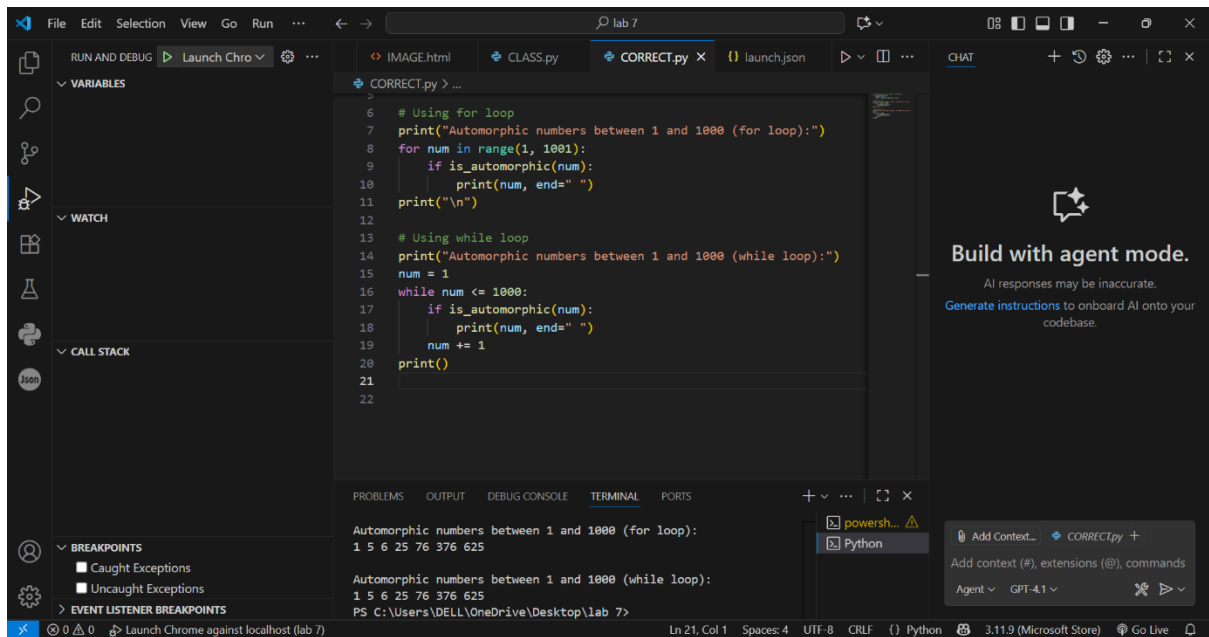
```
        print(num, end=" ")
```

```
    num += 1
```

```
print()
```



OUTPUT :



- **OBSERVATION : Correctness:** The logic for both loops directly applies the definition of automorphic number, checking if the square ends with the number itself.
- **Efficiency:** Both methods are efficient for this range and avoid unnecessary calculations. String comparison is appropriate here

Feature	For Loop	While Loop
Structure	Concise, ideal for ranged iteration	Flexible, allows custom increments
Readability	Slightly more straightforward	Slightly longer, but easy to follow
Performance	Identical (checks each number)	Identical (checks each number)

Task -3: 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.

PROMPT : 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.

CODE : def classify_feedback(rating):

```
    """
```

```
    Classify online shopping feedback based on numerical rating (1-5).
```

Args:

```
    rating (int): Rating value from 1 to 5.
```

Returns:

```
    str: Feedback category ("Positive", "Neutral", "Negative", or "Invalid Rating").
```

```
    """
```

```
    if 1 <= rating <= 5:
```

```
        if rating == 5:
```

```
            return "Positive"
```

```
        elif rating == 4 or rating == 3:
```

```
            return "Neutral"
```

```
        else: # rating is 1 or 2
```

```
            return "Negative"
```

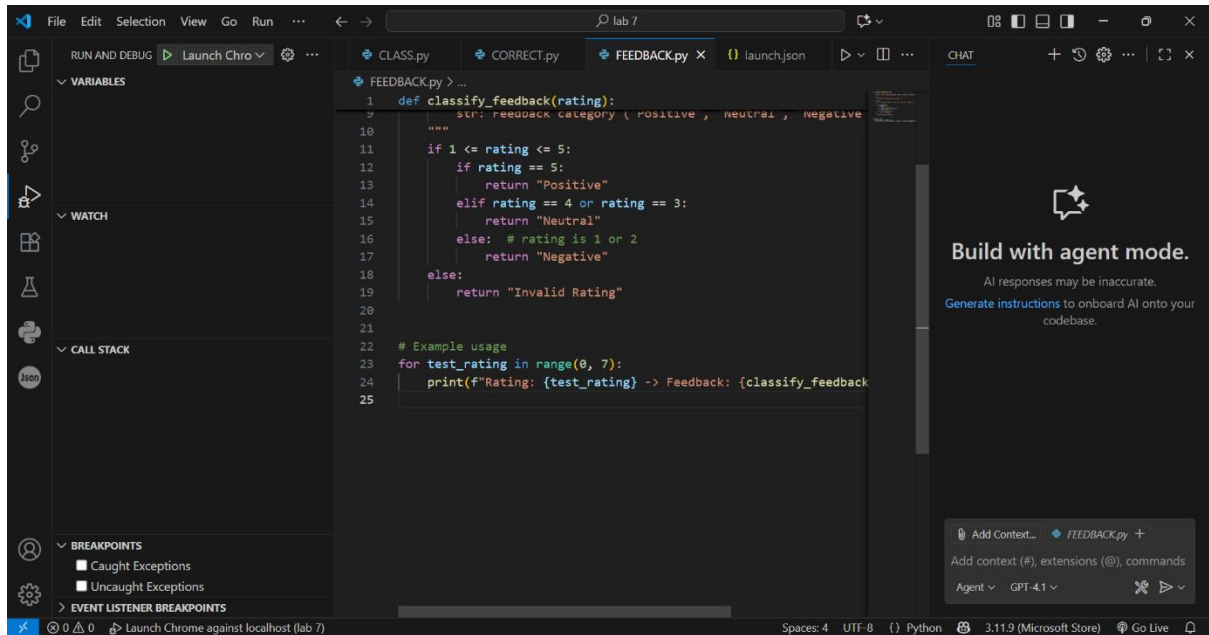
```
    else:
```

```
        return "Invalid Rating"
```

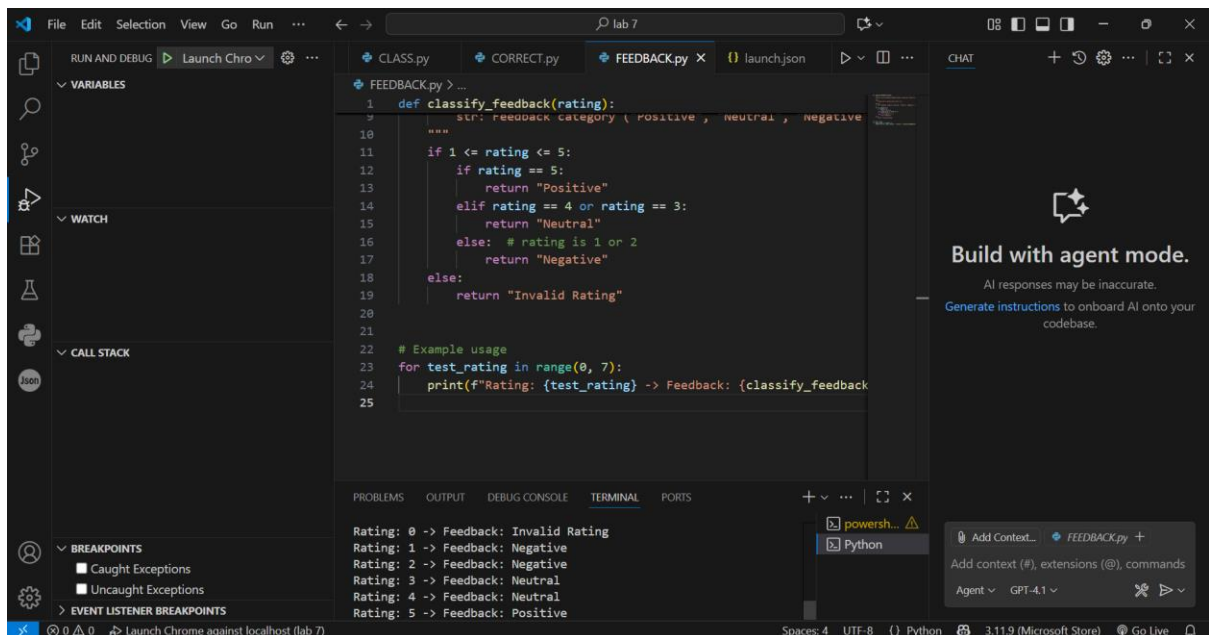
Example usage

for test_rating in range(0, 7):

print(f"Rating: {test_rating} -> Feedback: {classify_feedback(test_rating)}")



OUTPUT :



OBSERVATION : This code follows key clean code practices like meaningful naming, simple readable conditions, and documentation. It clearly separates valid from invalid inputs and groups ratings logically. The example usage demonstrates output for edge and normal cases.

Method	Correctness	Readability	Scalability	Efficiency
Nested if-elif-else	Accurate classification, handles all rating cases	Structured but can be verbose for many conditions	Less scalable with more conditions, nested blocks grow	Efficient for small ranges
Dictionary Mapping	Direct mapping from rating to feedback	Very clear and concise	Highly scalable, easy to add mappings	Fast dictionary lookup
match-case (Python 3.10+)	Matches multiple cases compactly	Very readable with grouped conditions	Scales well with multiple grouped cases	Efficient and clear branching

Task -4 : Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).

- Instructions:
 - o Get AI-generated code to list all primes using a for loop.
 - o Analyze the correctness and efficiency of the prime-checking logic.
 - o Ask AI to regenerate an optimized version (e.g., using the square root method).

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

- Instructions:
 - o Get AI-generated code to list all primes using a for loop.
 - o Analyze the correctness and efficiency of the prime-checking logic.
 - o Ask AI to regenerate an optimized version (e.g., using the square root method).

CODE : import math

```

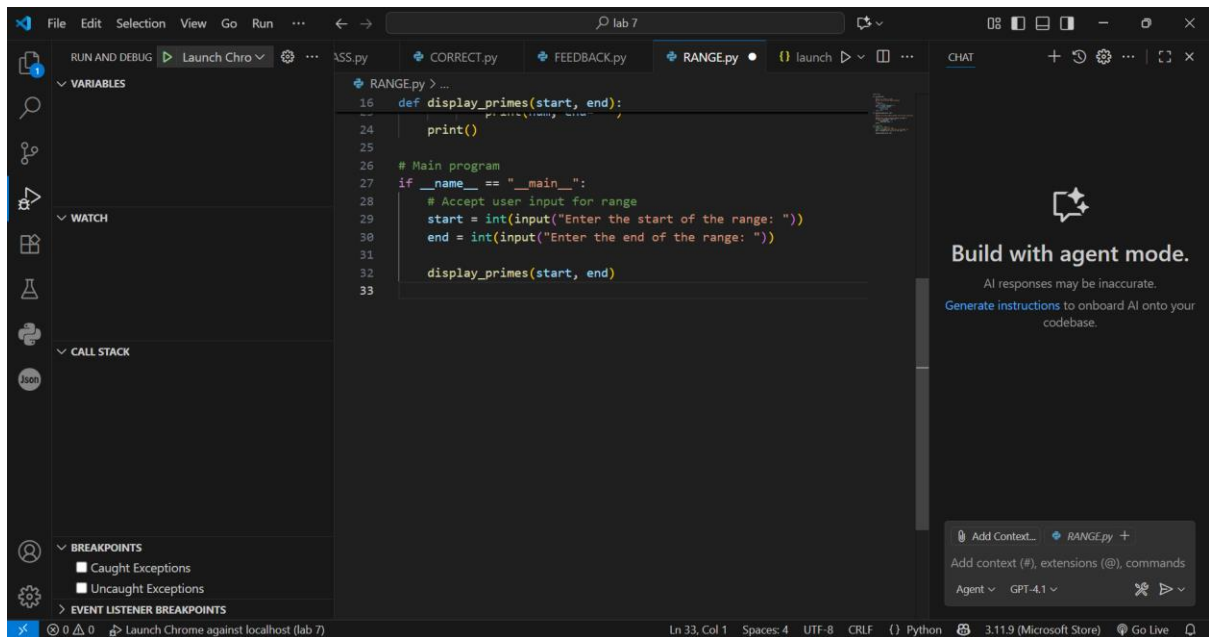
def is_prime(num):
    """
    Checks if a number is prime.
    Returns True if prime, False otherwise.
    """
    if num <= 1:
        return False
    limit = int(math.sqrt(num)) + 1
    for i in range(2, limit):
        if num % i == 0:
            return False
    return True

def display_primes(start, end):
    """
    Displays all prime numbers between start and end (inclusive).
    """
    print(f"Prime numbers between {start} and {end}:")
    for num in range(start, end + 1):
        if is_prime(num):
            print(num, end=" ")
    print()

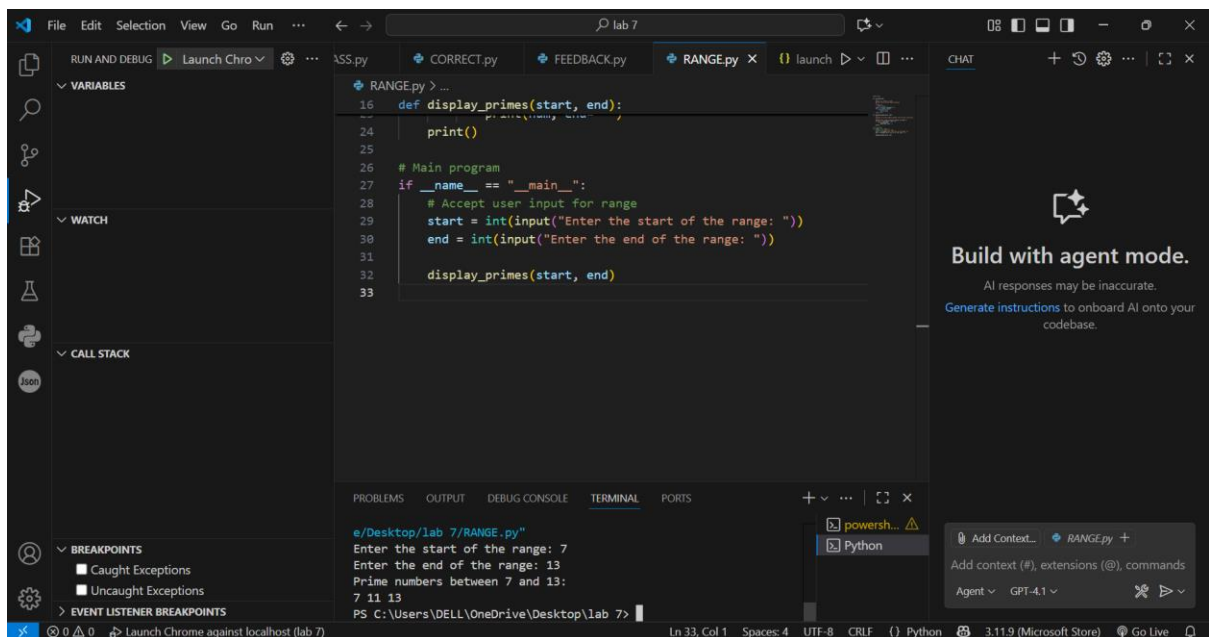
# Main program
if __name__ == "__main__":
    # Accept user input for range
    start = int(input("Enter the start of the range: "))
    end = int(input("Enter the end of the range: "))

    display_primes(start, end)

```



OUTPUT :



- **OBSERVATION :** The `is_prime` function checks primality by testing divisibility up to the square root of the number, which is efficient.
- The `display_primes` function iterates through the specified range and prints primes found.
- User inputs the start and end values, making the program flexible for any range.

Task -5 : 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

PROMPT : 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

CODE : class Library:

```
def __init__(self):
    """
    Initialize the Library with an empty dictionary to store books.
    Key: book title (str), Value: availability (bool)
    """
    self.books = {}

def add_book(self, title):
    """
    Add a new book to the library.
    If the book already exists, notify the user.
    """
    if title in self.books:
        print(f'Book "{title}" already exists in the library.')
    else:
        self.books[title] = True # True means book is available
        print(f'Book "{title}" added to the library.')
```

```
def issue_book(self, title):
    """
    Issue a book if available.
    Handles edge cases:
    - Book does not exist
    - Book already issued
    """
    if title not in self.books:
        print(f'Book "{title}" does not exist in the library.')
    elif not self.books[title]:
        print(f'Book "{title}" is already issued.')
    else:
        self.books[title] = False
        print(f'Book "{title}" has been issued.')
```

```
def display_books(self):
    """
    Display all books with their availability status.
    """
    if not self.books:
        print("Library has no books.")
        return
    print("Available books in the library:")
    for title, available in self.books.items():
        status = "Available" if available else "Issued"
        print(f'- {title} [{status}]')
```

Example usage:

```
library = Library()
```

```
library.add_book("The Catcher in the Rye")
```

```
library.add_book("To Kill a Mockingbird")
```

```
library.add_book("1984")
```

```
library.display_books()
```

```
print()
```

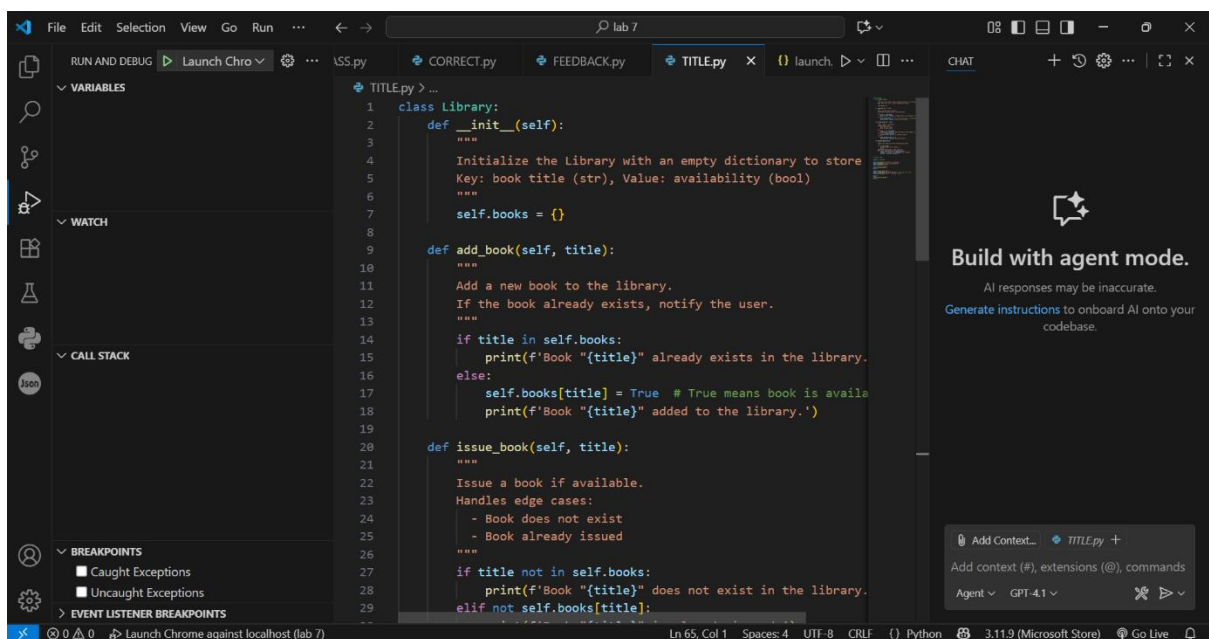
```
library.issue_book("1984")
```

```
library.issue_book("The Great Gatsby") # Book does not exist
```

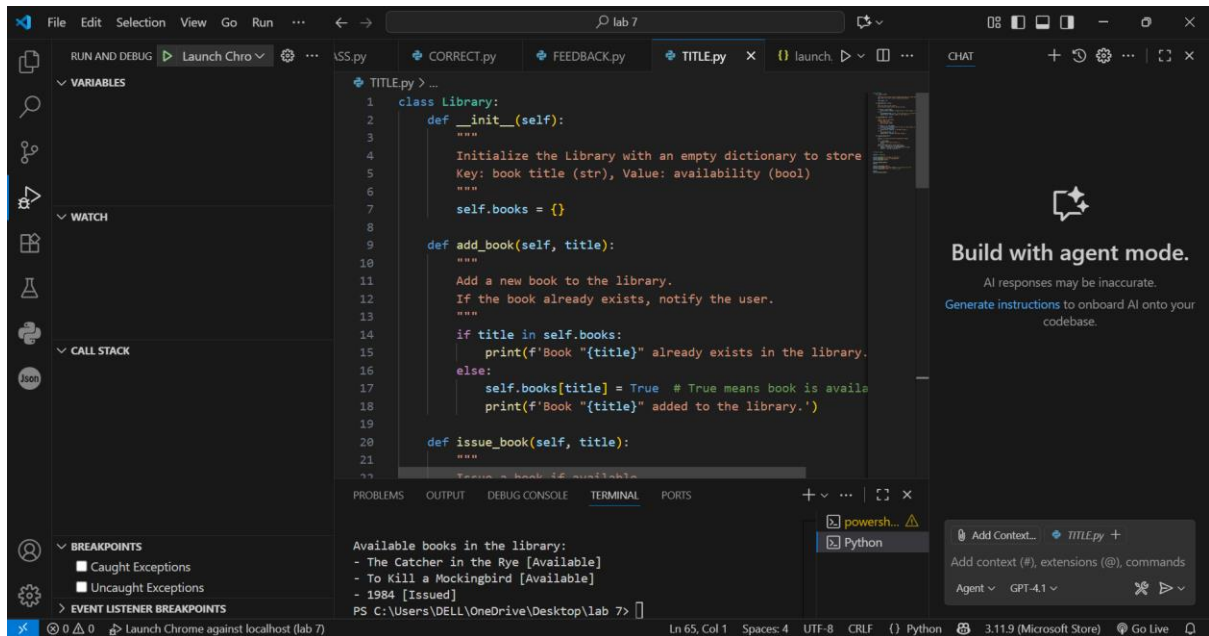
```
library.issue_book("1984") # Already issued
```

```
print()
```

```
library.display_books()
```



OUTPUT :



```
1 class Library:
2     def __init__(self):
3         """
4         Initialize the Library with an empty dictionary to store
5         Key: book title (str), Value: availability (bool)
6         """
7         self.books = {}
8
9     def add_book(self, title):
10        """
11        Add a new book to the library.
12        If the book already exists, notify the user.
13        """
14        if title in self.books:
15            print(f'Book "{title}" already exists in the library.')
16        else:
17            self.books[title] = True # True means book is available
18            print(f'Book "{title}" added to the library.')
19
20    def issue_book(self, title):
21        """
22        Issue a book if available
23        """
```

Available books in the library:

- The Catcher in the Rye [Available]
- To Kill a Mockingbird [Available]
- 1984 [Issued]

PS C:\Users\DELL\OneDrive\Desktop\lab 7>

- OBSERVATION : The add_book() method adds books with an "issued" status set to False initially.
- The issue_book() method checks if the book exists and is not already issued before issuing; it handles edge cases gracefully by printing appropriate messages.
- The display_books() method lists all books with their availability.
- Case-insensitive matching is used to improve usability.

This class and usage example demonstrate robust handling of typical library management actions, with clear comments and readable structure.