ASSIGNMENT – 7.3

2303A51355

Batch-10

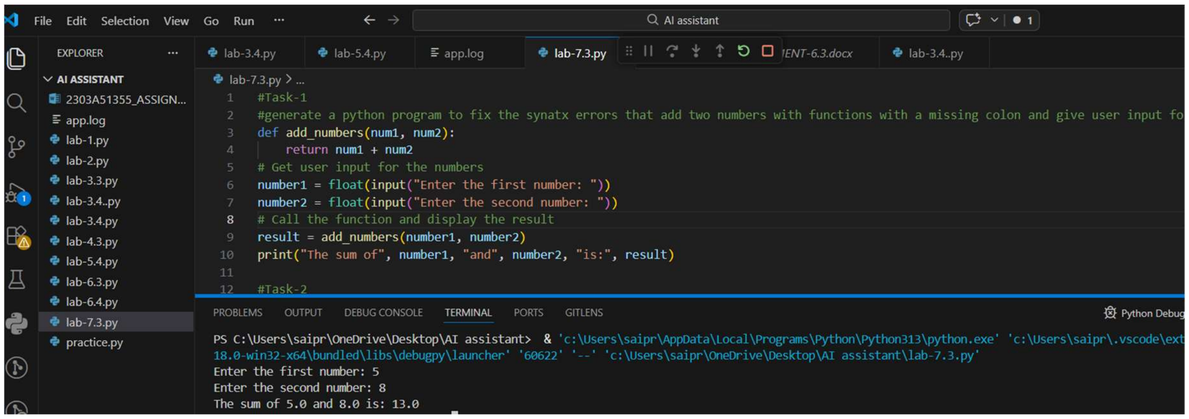
Task-1

Prompt: generate a python program to fix the synatx errors that add two numbers with functions with a missing colon and give user input for the numbers.

code :

```
def add_numbers(num1, num2):
    return num1 + num2
# Get user input for the numbers
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
# Call the function and display the result
result = add_numbers(number1, number2)
print("The sum of", number1, "and", number2, "is:", result)
```

Output :



```
#Task-1
#generate a python program to fix the synatx errors that add two numbers with functions with a missing colon and give user input fo
def add_numbers(num1, num2):
    return num1 + num2
# Get user input for the numbers
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
# Call the function and display the result
result = add_numbers(number1, number2)
print("The sum of", number1, "and", number2, "is:", result)

#Task-2
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\saipr\.vscode\ext
18.0-win32-x64\bundled\libs\debugpy\launcher' '60622' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assistant\lab-7.3.py'
Enter the first number: 5
Enter the second number: 8
The sum of 5.0 and 8.0 is: 13.0
```

Code Analysis:

- The function add_numbers() takes two parameters and returns their sum.
- The missing colon after the function definition is corrected.

- User inputs are converted to float to allow decimal values.
- The function is called with user inputs and the result is printed.
- Using functions improves reusability and modular programming.

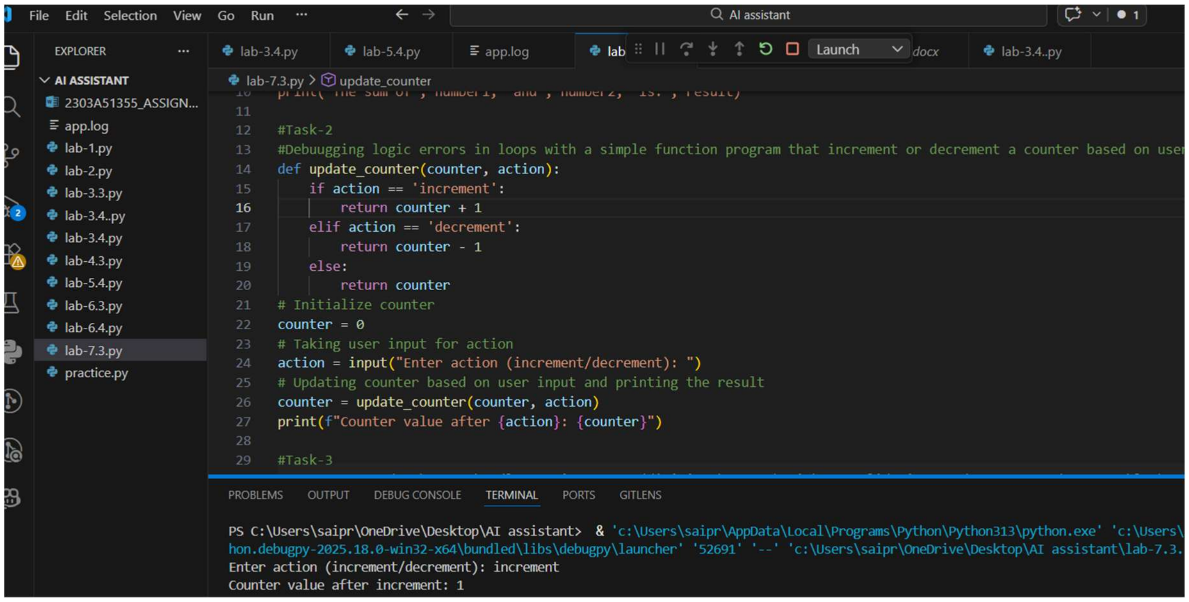
Task-2

Prompt: Debugging logic errors in loops with a simple function program that increment or decrement a counter based on user input.

Code :

```
def update_counter(counter, action):
    if action == 'increment':
        return counter + 1
    elif action == 'decrement':
        return counter - 1
    else:
        return counter
# Initialize counter
counter = 0
# Taking user input for action
action = input("Enter action (increment/decrement):
")
# Updating counter based on user input and printing
the result
counter = update_counter(counter, action)
print(f"Counter value after {action}: {counter}")
```

Output :



```
print("The sum of", number1, "and", number2, "is:", result)

#Task-2
#Debuugging logic errors in loops with a simple function program that increment or decrement a counter based on use
def update_counter(counter, action):
    if action == 'increment':
        return counter + 1
    elif action == 'decrement':
        return counter - 1
    else:
        return counter
# Initialize counter
counter = 0
# Taking user input for action
action = input("Enter action (increment/decrement): ")
# Updating counter based on user input and printing the result
counter = update_counter(counter, action)
print(f"Counter value after {action}: {counter}")

#Task-3
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52691' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assistant\lab-7.3.
Enter action (increment/decrement): increment
Counter value after increment: 1
```

Code Analysis:

- The function modifies the counter based on user action.
- action.lower() avoids case-sensitivity issues.
- If invalid input is entered, the counter remains unchanged.
- The logic ensures proper increment/decrement functionality.
- This demonstrates basic debugging of logical conditions.

Task-3

Prompt: generate a code that to handle runtime errors(division by zero) without validations and use try and except blocks to catch the error. take user input with functions

Code :

```
def divide_numbers(num1, num2):
    try:
        result = num1 / num2
        return result
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
# Get user input for the numbers
```

number1 = float(input("Enter the numerator: "))

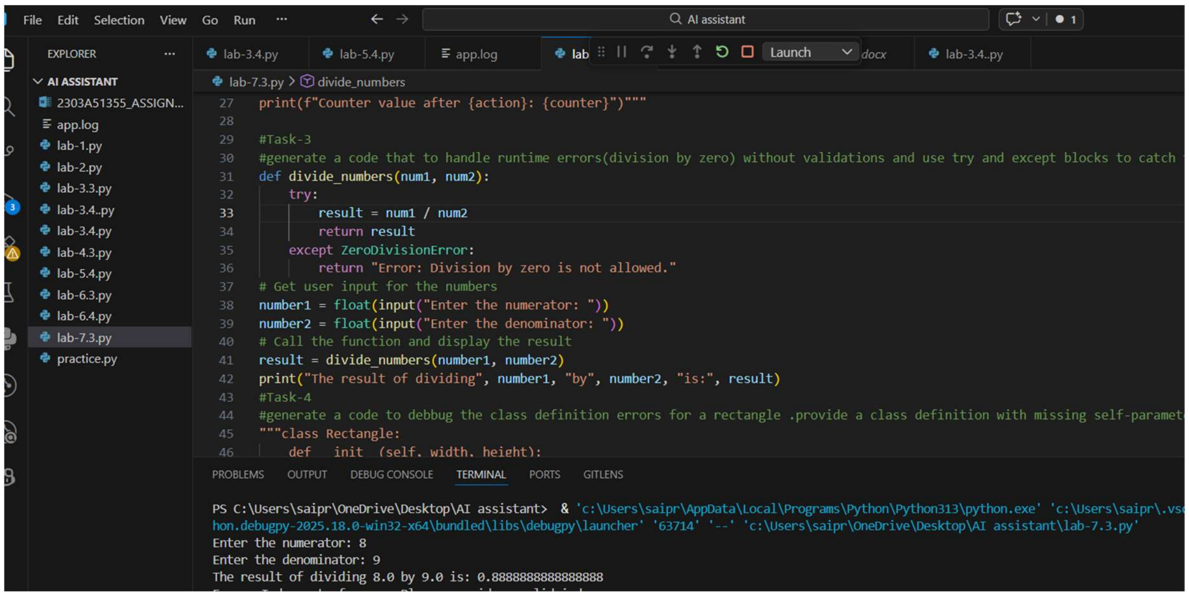
number2 = float(input("Enter the denominator: "))

Call the function and display the result

result = divide_numbers(number1, number2)

print("The result of dividing", number1, "by", number2, "is:", result)

Output:



```python
print(f"Counter value after {action}: {counter}")"""

#Task-3
#generate a code that to handle runtime errors(division by zero) without validations and use try and except blocks to catch
def divide_numbers(num1, num2):
    try:
        result = num1 / num2
        return result
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
# Get user input for the numbers
number1 = float(input("Enter the numerator: "))
number2 = float(input("Enter the denominator: "))
# Call the function and display the result
result = divide_numbers(number1, number2)
print("The result of dividing", number1, "by", number2, "is:", result)
#Task-4
#generate a code to debbug the class definition errors for a rectangle .provide a class definition with missing self-paramet
"""class Rectangle:
    def __init__(self, width, height):
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\saipr\.vs
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63714' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assistant\lab-7.3.py'
Enter the numerator: 8
Enter the denominator: 9
The result of dividing 8.0 by 9.0 is: 0.8888888888888888
```

Code Analysis :

- The function attempts division inside a try block.
- If the denominator is zero, ZeroDivisionError is caught.
- The program does not crash due to exception handling.
- A user-friendly error message is returned instead.
- try-except ensures runtime stability.

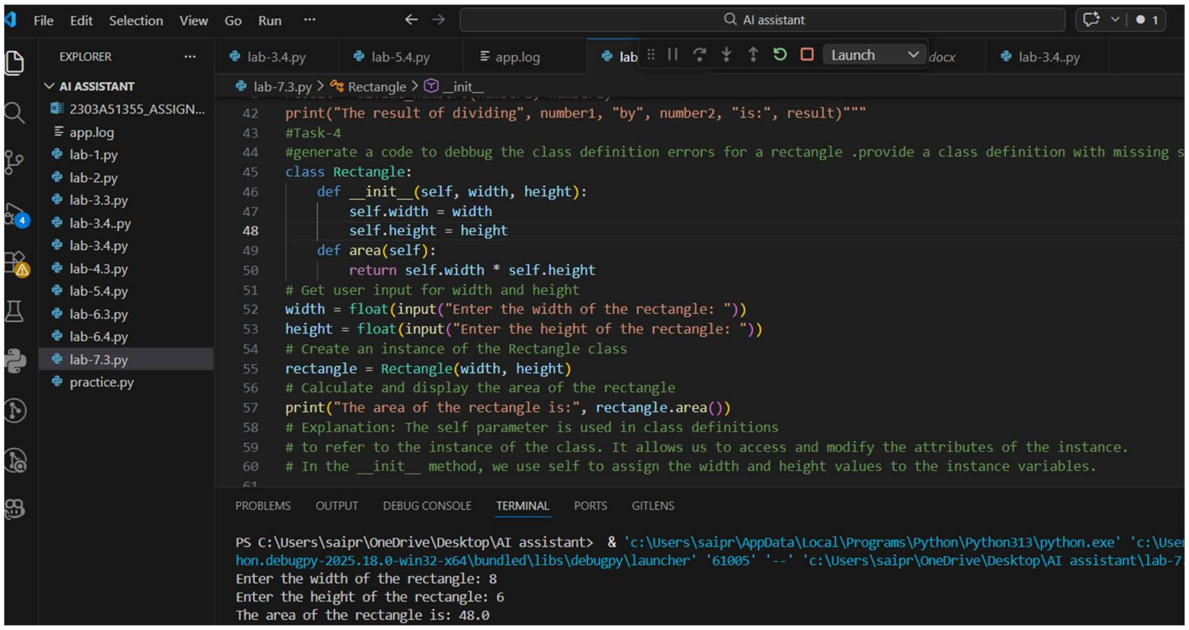
Task-4

Prompt: #generate a code to debug the class definition errors for a rectangle .provide a class definition with missing self-parameter and correct it using __init_ method and explain why self is used in class definitions .take user input

Code :

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height
# Get user input for width and height
width = float(input("Enter the width of the rectangle: "))
height = float(input("Enter the height of the rectangle: "))
# Create an instance of the Rectangle class
rectangle = Rectangle(width, height)
# Calculate and display the area of the rectangle
print("The area of the rectangle is:", rectangle.area())
# Explanation: The self parameter is used in class definitions
# to refer to the instance of the class. It allows us to access and
modify the attributes of the instance.
# In the __init__ method, we use self to assign the width and
height values to the instance variables.
```

Output:



```python
print("The result of dividing", number1, "by", number2, "is:", result)"""
#Task-4
#generate a code to debbug the class definition errors for a rectangle .provide a class definition with missing s
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height
# Get user input for width and height
width = float(input("Enter the width of the rectangle: "))
height = float(input("Enter the height of the rectangle: "))
# Create an instance of the Rectangle class
rectangle = Rectangle(width, height)
# Calculate and display the area of the rectangle
print("The area of the rectangle is:", rectangle.area())
# Explanation: The self parameter is used in class definitions
# to refer to the instance of the class. It allows us to access and modify the attributes of the instance.
# In the __init__ method, we use self to assign the width and height values to the instance variables.
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Use
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61005' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assistant\lab-7
Enter the width of the rectangle: 8
Enter the height of the rectangle: 6
The area of the rectangle is: 48.0
```

Code Analysis :

- The constructor method must be __init__ (double underscores).
- self refers to the current object instance.
- Instance variables (self.width, self.height) store object data.
- The area() method accesses instance variables using self.
- Without self, Python cannot link data to the specific object.

Task-5

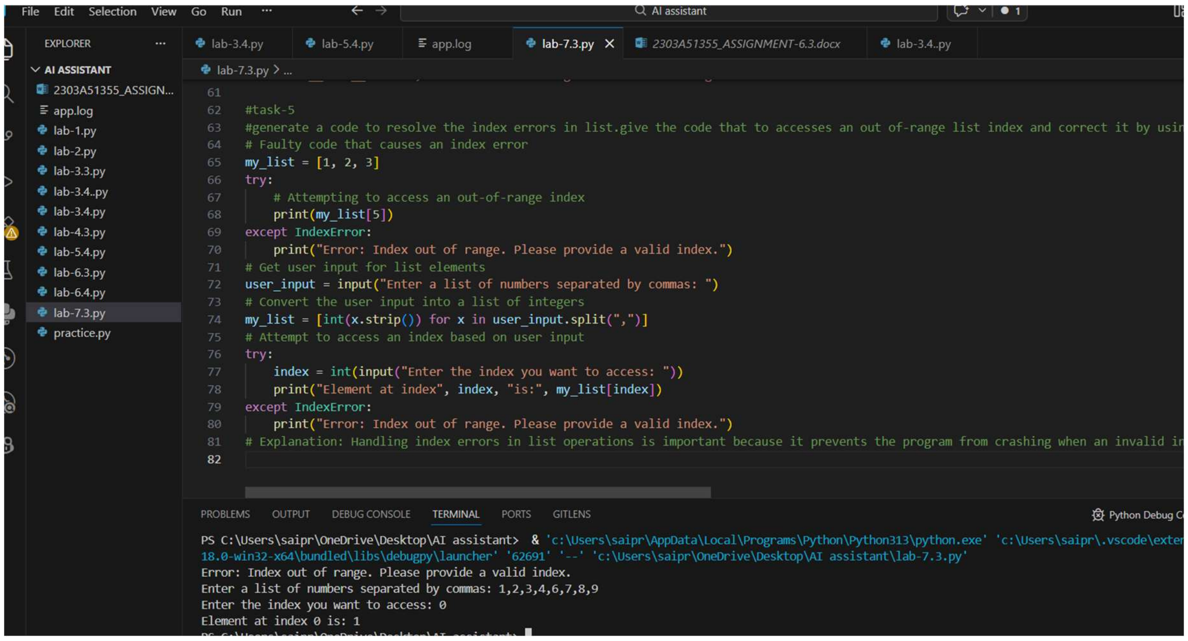
Prompt: generate a code to resolve the index errors in list.give the code that to accesses an out of-range list index and correct it by using exception handling and explain the importance of handling index errors in list operations. take user input for list elements.

```
my_list = [1, 2, 3]

try:

    # Attempting to access an out-of-range index

    print(my_list[5])

except IndexError:

    print("Error: Index out of range. Please provide a valid index.")
```

```python
# Get user input for list elements
user_input = input("Enter a list of numbers separated by commas: ")
# Convert the user input into a list of integers
my_list = [int(x.strip()) for x in user_input.split(",")]
# Attempt to access an index based on user input
try:
    index = int(input("Enter the index you want to access: "))
    print("Element at index", index, "is:", my_list[index])
except IndexError:
    print("Error: Index out of range. Please provide a valid index.")
```

Explanation: Handling index errors in list operations is important because it prevents the program from crashing when an invalid index is accessed. By using exception handling, we can catch the error and provide a user-friendly message, allowing the program to continue running smoothly even when unexpected input is encountered.

Output :



```python

#task-5
#generate a code to resolve the index errors in list.give the code that to accesses an out of-range list index and correct it by usin
# Faulty code that causes an index error
my_list = [1, 2, 3]
try:
    # Attempting to access an out-of-range index
    print(my_list[5])
except IndexError:
    print("Error: Index out of range. Please provide a valid index.")
# Get user input for list elements
user_input = input("Enter a list of numbers separated by commas: ")
# Convert the user input into a list of integers
my_list = [int(x.strip()) for x in user_input.split(",")]
# Attempt to access an index based on user input
try:
    index = int(input("Enter the index you want to access: "))
    print("Element at index", index, "is:", my_list[index])
except IndexError:
    print("Error: Index out of range. Please provide a valid index.")
# Explanation: Handling index errors in list operations is important because it prevents the program from crashing when an invalid in
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\saipr\.vscode\exter
18.0-win32-x64\bundled\libs\debugpy\launcher' '62691' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assistant\lab-7.3.py'
Error: Index out of range. Please provide a valid index.
Enter a list of numbers separated by commas: 1,2,3,4,6,7,8,9
Enter the index you want to access: 0
Element at index 0 is: 1
```

Code Analysis :

- User input is converted into a list using split() and list comprehension.
- The program attempts to access a user-specified index.
- If index is invalid, IndexError is handled gracefully.
- ValueError ensures proper numeric input.
- Exception handling prevents program crashes and improves reliability