

ASSIGNMENT – 7.3

2303A51525

Batch-10

Task-1

Prompt: generate a python program to fix the syntax 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 :

```
1 #generate a python program to fix the syntax errors that add two numbers with functions with a missing colon and give
2 def add_numbers(num1, num2):
3     return num1 + num2
4 # Get user input for the numbers
5 number1 = float(input("Enter the first number: "))
6 number2 = float(input("Enter the second number: "))
7 # Call the function to add the numbers and print the result
8 result = add_numbers(number1, number2)
9 print("The sum of the two numbers is:", result)
```

```
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-7.3.py
Enter the first number: 3
Enter the second number: 6
The sum of the two numbers is: 9.0
PS C:\Users\hp\OneDrive\Desktop\AI>
```

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

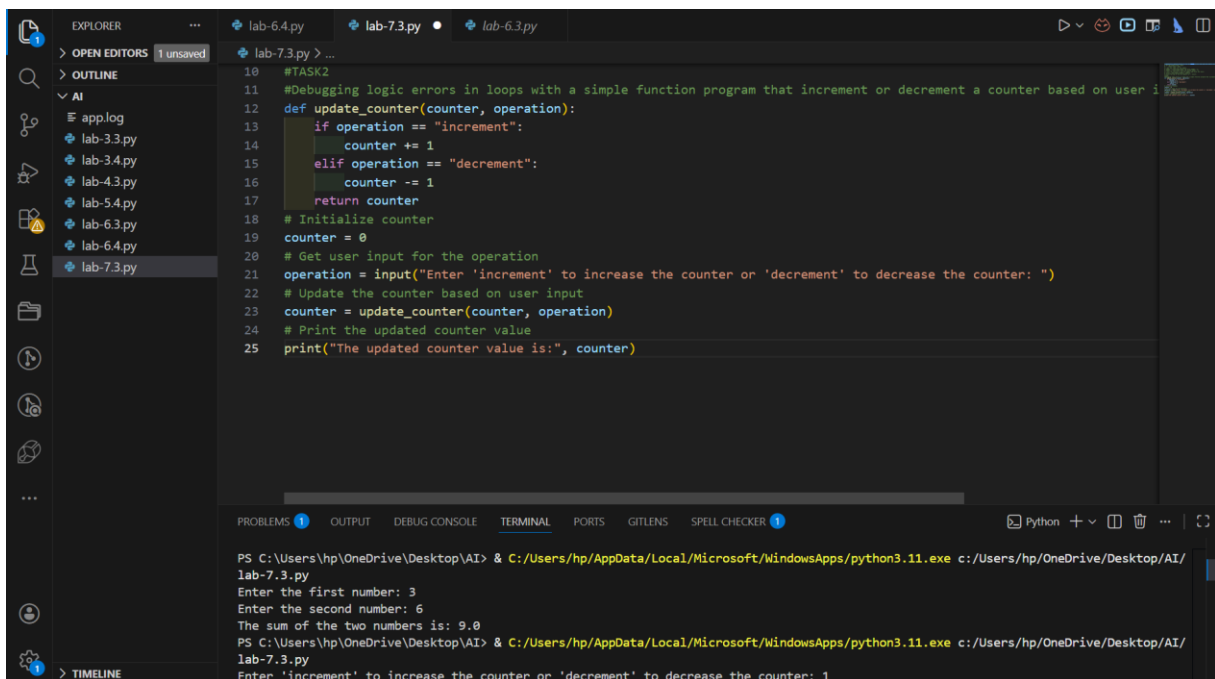
# Taking user input for action action = input("Enter
action (increment/decrement): ")

# Updating counter based on user input and printing
the result counter = update_counter(counter, action)

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

```

Output :



The screenshot shows a Visual Studio Code editor with a Python file named 'lab-7.3.py' open. The code defines a function 'update_counter' that takes a counter and an operation as input. It uses an if-elif structure to increment or decrement the counter based on the operation. The main code initializes the counter to 0, gets user input for the operation, calls the 'update_counter' function, and prints the result. The terminal at the bottom shows the command to run the script and the output, which includes the user's input and the resulting counter value.

```

10 #TASK2
11 #Debugging logic errors in loops with a simple function program that increment or decrement a counter based on user input
12 def update_counter(counter, operation):
13     if operation == "increment":
14         counter += 1
15     elif operation == "decrement":
16         counter -= 1
17     return counter
18 # Initialize counter
19 counter = 0
20 # Get user input for the operation
21 operation = input("Enter 'increment' to increase the counter or 'decrement' to decrease the counter: ")
22 # Update the counter based on user input
23 counter = update_counter(counter, operation)
24 # Print the updated counter value
25 print("The updated counter value is:", counter)

```

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-7.3.py
Enter the first number: 3
Enter the second number: 6
The sum of the two numbers is: 9.0
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-7.3.py
Enter 'increment' to increase the counter or 'decrement' to decrease the counter: 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:

Code Analysis :

```
26 #task3
27 #generate a code that to handle runtime errors(dvision by zero) without validations and use try and except blocks to
28 def divide_numbers(num1, num2):
29     try:
30         result = num1 / num2
31         return result
32     except ZeroDivisionError:
33         return "Error: Cannot divide by zero."
34 # Get use (function) def divide_numbers(
35 number1 = num1: Any,
36 number2 = num2: Any
37 # Call th ) -> (Any | Literal['Error: Cannot divide by zero.'])
38 result = divide_numbers(number1, number2)
39 print("The result of the division is:", result)
```

```
Enter 'increment' to increase the counter or 'decrement' to decrease the counter: 1
The updated counter value is: 0
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/
lab-7.3.py
Enter the first number: 9
Enter the second number: 8
The result of the division is: 1.125
PS C:\Users\hp\OneDrive\Desktop\AI>
```

- ☐ 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

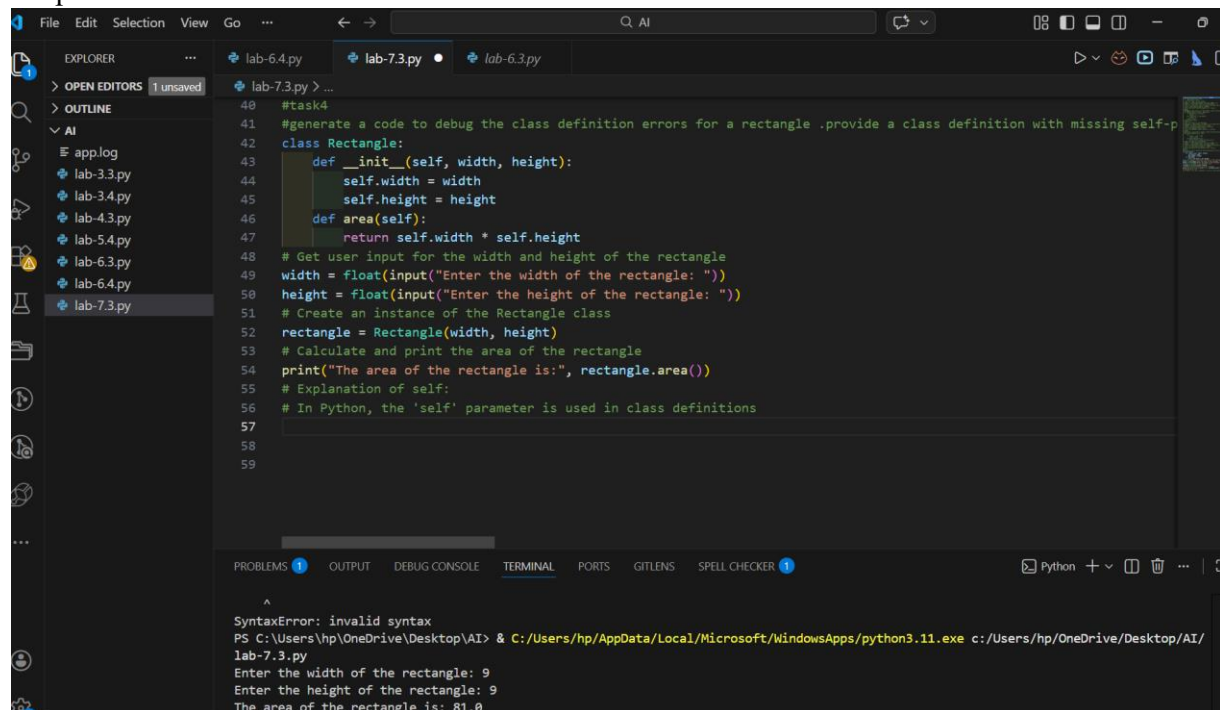
width and height width = float(input("Enter the width of
the rectangle: ")) height = float(input("Enter the height
of the rectangle: ")) # Create an instance of the Rectangle
class rectangle = Rectangle(width, height) # Calculate
and display the area of the rectangle print("The area of
the rectangle 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:



The screenshot shows a Visual Studio Code editor with a Python file named `lab-7.3.py` open. The code defines a `Rectangle` class with an `__init__` method and an `area` method. It then creates an instance of the class, calculates the area, and prints it. The terminal at the bottom shows the execution of the script, which prompts for the width and height of the rectangle and outputs the calculated area.

```

40 #task4
41 #generate a code to debug the class definition errors for a rectangle .provide a class definition with missing self-p
42 class Rectangle:
43     def __init__(self, width, height):
44         self.width = width
45         self.height = height
46     def area(self):
47         return self.width * self.height
48 # Get user input for the width and height of the rectangle
49 width = float(input("Enter the width of the rectangle: "))
50 height = float(input("Enter the height of the rectangle: "))
51 # Create an instance of the Rectangle class
52 rectangle = Rectangle(width, height)
53 # Calculate and print the area of the rectangle
54 print("The area of the rectangle is:", rectangle.area())
55 # Explanation of self:
56 # In Python, the 'self' parameter is used in class definitions
57
58
59

```

```

^
SyntaxError: invalid syntax
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/
lab-7.3.py
Enter the width of the rectangle: 9
Enter the height of the rectangle: 9
The area of the rectangle is: 81.0

```

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:

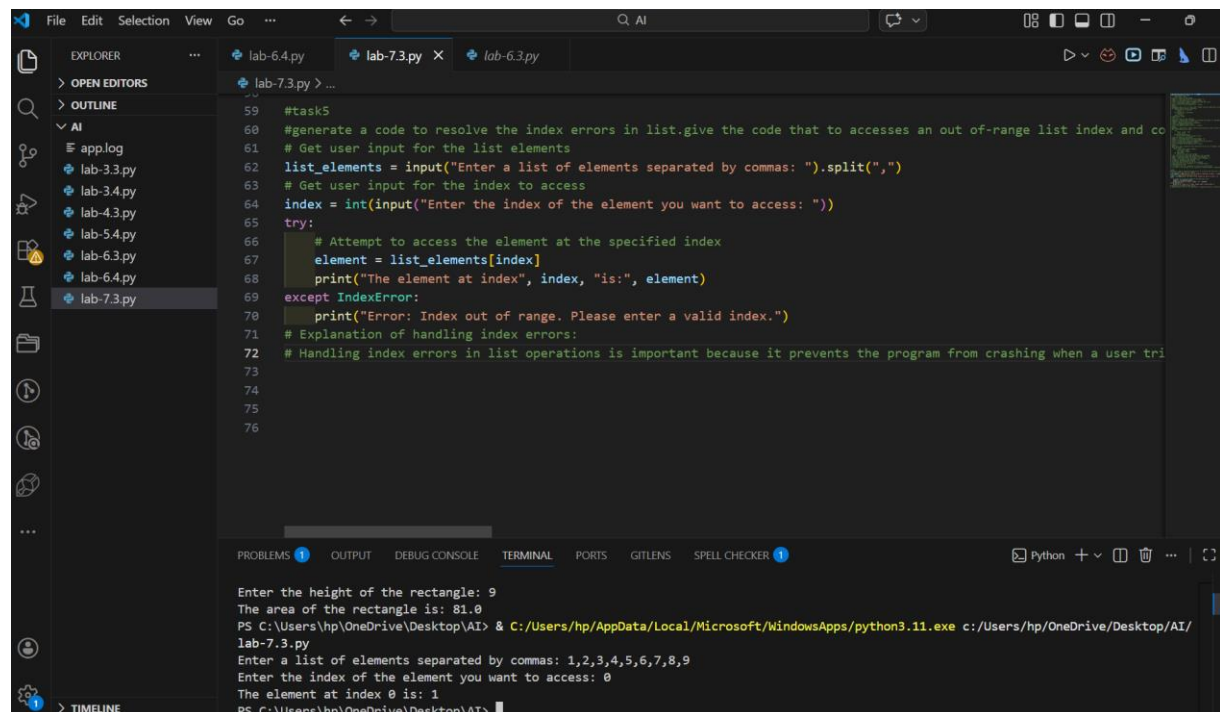
    print("Error: Index out of range. Please provide a valid index.")
# Get user input for list elements user_input = input("Enter a list of
numbers separated by commas: ")
# Convert the user input into a list of integers my_list
= [int(x.strip()) for x in user_input.split(",")] #
Attempt to access an index 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 :



The screenshot shows a Visual Studio Code editor with a Python file named `lab-7.3.py` open. The code is a script for handling index errors in a list. It prompts the user to enter a list of elements separated by commas and then an index to access. It uses a `try-except` block to catch `IndexError` and print a user-friendly message.

```
59 #task5
60 #generate a code to resolve the index errors in list.give the code that to accesses an out of-range list index and co
61 # Get user input for the list elements
62 list_elements = input("Enter a list of elements separated by commas: ").split(",")
63 # Get user input for the index to access
64 index = int(input("Enter the index of the element you want to access: "))
65 try:
66     # Attempt to access the element at the specified index
67     element = list_elements[index]
68     print("The element at index", index, "is:", element)
69 except IndexError:
70     print("Error: Index out of range. Please enter a valid index.")
71 # Explanation of handling index errors:
72 # Handling index errors in list operations is important because it prevents the program from crashing when a user tri
73
74
75
76
```

The terminal output shows the execution of the script:

```
Enter the height of the rectangle: 9
The area of the rectangle is: 81.0
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/
lab-7.3.py
Enter a list of elements separated by commas: 1,2,3,4,5,6,7,8,9
Enter the index of the element you want to access: 0
The element at index 0 is: 1
PS C:\Users\hp\OneDrive\Desktop\AI>
```

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