

## ASSIGNMENT – 7.3

2303A51355

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.

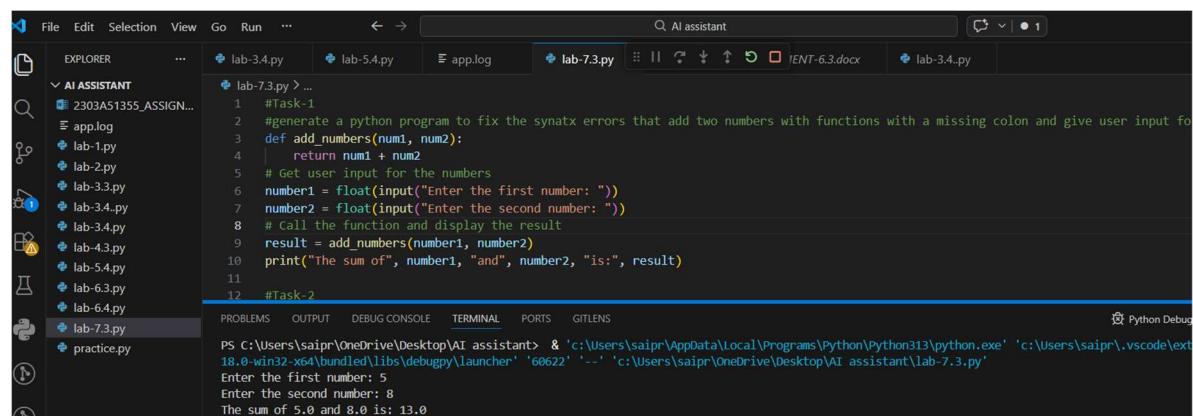
code :

```
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 :



```
File Edit Selection View Go Run ... AI assistant
EXPLORER ... lab-3.4.py lab-5.4.py app.log lab-7.3.py lab-3.4.py
AI ASSISTANT 2303A51355_ASSIGN...
lab-1.py
lab-2.py
lab-3.py
lab-3.3.py
lab-3.4.py
lab-3.4.py
lab-4.3.py
lab-5.4.py
lab-6.3.py
lab-6.4.py
lab-7.3.py
practice.py
lab-7.3.py > ...
1 #Task-1
2 #generate a python program to fix the syntax errors that add two numbers with functions with a missing colon and give user input fo
3 def add_numbers(num1, num2):
4     return num1 + num2
5 # Get user input for the numbers
6 number1 = float(input("Enter the first number: "))
7 number2 = float(input("Enter the second number: "))
8 # call the function and display the result
9 result = add_numbers(number1, number2)
10 print("The sum of", number1, "and", number2, "is:", result)
11
12 #Task-2
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
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 :

```

File Edit Selection View Go Run ... ← → Q AI assistant
EXPLORER ... lab-3.4.py lab-5.4.py app.log lab docx lab-3.4.py
AI ASSISTANT 2303A51355_ASSIGN...
app.log
lab-1.py
lab-2.py
lab-3.3.py
lab-3.4.py
lab-3.4.py
lab-4.3.py
lab-5.4.py
lab-6.3.py
lab-6.4.py
lab-7.3.py
practice.py
lab-7.3.py > update_counter
10     print(f'The sum of {number1} and {number2} is: {result}')
11
12 #Task-2
13 #Debugging logic errors in loops with a simple function program that increment or decrement a counter based on user
14 def update_counter(counter, action):
15     if action == 'increment':
16         return counter + 1
17     elif action == 'decrement':
18         return counter - 1
19     else:
20         return counter
21 # Initialize counter
22 counter = 0
23 # Taking user input for action
24 action = input("Enter action (increment/decrement): ")
25 # Updating counter based on user input and printing the result
26 counter = update_counter(counter, action)
27 print(f"Counter value after {action}: {counter}")
28
29 #Task-3

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```

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:

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** On the left, there is a tree view of files. The file `lab-7.3.py` is currently selected.
- Code Editor:** The main area displays Python code for a function `divide_numbers` that handles division by zero using a try-except block. It also demonstrates how to get user input for the numerator and denominator and print the result.
- Terminal:** At the bottom, the terminal window shows the command used to run the script and the output of the program asking for user input.

```
File Edit Selection View Go Run ... < > Q AI assistant

EXPLORER ... lab-3.4.py lab-5.4.py app.log lab || Launch docx lab-3.4.py

AI ASSISTANT
2303AS1355_ASSIGN...
app.log
lab-1.py
lab-2.py
lab-3.py
lab-3.4.py
lab-3.4.py
lab-4.py
lab-4.3.py
lab-5.4.py
lab-6.3.py
lab-6.4.py
lab-7.3.py
practice.py

lab-7.3.py > divide_numbers

27     print(f"Counter value after {action}: {counter}")
28
29 #Task-3
30 #generate a code that to handle runtime errors(division by zero) without validations and use try and except blocks to catch
31 def divide_numbers(num1, num2):
32     try:
33         result = num1 / num2
34         return result
35     except ZeroDivisionError:
36         return "Error: Division by zero is not allowed."
37 # Get user input for the numbers
38 number1 = float(input("Enter the numerator: "))
39 number2 = float(input("Enter the denominator: "))
40 # Call the function and display the result
41 result = divide_numbers(number1, number2)
42 print("The result of dividing", number1, "by", number2, "is:", result)
43 #Task-4
44 #generate a code to debug the class definition errors for a rectangle .provide a class definition with missing self-paramet
45 """class Rectangle:
46     def __init__(self, width, height):
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\saipr\vscode.debugpy-2023.18.0-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:

## 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 the Visual Studio Code interface. The code editor displays the Python script above. The terminal at the bottom shows the execution of the script and its output:

```

C:\Users\saipr\OneDrive\Desktop\AI assistant> & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\saipr\vscode\exten
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